THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 2 Vol. 1, No. 2 $2.50

File Transfer Programs for CP/M page 2

Part Two of a Series:

The RS-232-C Serial Interface page 6

Part One:

Build a Hardware Print Spooler page 12

A Review of Floppy Disk Formats page 14

Sending Morse Code With an Apple][page 16

Beginner's Column, Part Two':

Anyone For a Little "KISS" Electronics? page 19

WHAT IS A HACKER??

The September fifth issue of Newsweek contains a six page article "Beware: Hackers at Play," with a cover picture and the headline "Computer Capers." Several months ago, Wall Street Journal also ran a front page article about how Hackers break into computer systems.

I consider myself a Hacker, but I have no interest in breaking into computers (it takes more time than I have available to try to figure out what's going on in my own computer). I object to the fact that the press has defined hacking as breaking into computer systems. The press is giving all hackers a bad name because of the trespasses of a very few who call themselves Hackers. When I mention computer hacking, people ask 'How many computers have you broken into today?' I tell them that this is not hacking; but when they ask what hacking is, I have a hard time trying to explain it.

And so I put the question to our readers, 'What is Hacking?' What is it that Hackers do? We at The Computer Hacker would like to generate some good press about hacking, and need ideas and information from our readers. We will use this information to prepare press releases and an information packet for the press.

In order to get the information needed,The Computer Hacker is announcing a contest for Hackers. We'll award some prizes (perhaps logic probes or breadboarding kits) for the responses which are most useful. There will be two categories:

1) What is a Hacker?

2) A description of a useful hacker project, such as interfacing a micro so that a disabled person can control their world.

The final result, which will be submitted to the press, will probably be a blending of many responses. WE NEED YOUR INPUT!!

THE "HACKER STANDARD INTERFACE"

In this issue, we continue the series on the RS-232-C interface. The first part explained the standard's definitions. Part two covers the standart configurations and describes real world examples of the RS-232-C interface as used with microcomputers, and recommenations for hacker standards. A future article will cover the use of UART and USART integrated circuits with the RS-232-C serial interface. It may seem that we are spending a lot of time on the RE-232-C interface, but many hacker projects (at least the ones I get involved with) require the transfer of data, and a good understanding of the standard interfaces is vital. We suggest that most hacker projects be built as separate stand-alone devices using a standard interface instead of being built to operate only with a specific computer.

The problem with a computer specific peripheral is that the device will probably not work with another computer. Computer technology is changing very rapidly, and most of us will eventually get a different (or an additional) computer which will not be able to use the same computer specific peripherals. There is also the possibility that you will want to lend (or sell) your project to someone with a different computer. The decision to use either a computer specific or a standard interface design is not always clear-cut. There are advantages and disadvantages to both approaches, and you'll have to decide on a case-by-case basis. When you do design something which is not meant to be computer specific, you should use a standard interface if at all possible.

In future issues we will continue coverage of the RS-232-C interface, and will also cover other standard interfaces suitable for the hacker. We are interested in feedback from the field, so please tell us about your experiences with interfaces (both the good and the bad). If you would like to write an article, perhaps something on A/D and D/A or the factors to consider when deciding between using a computer specific versus a standard interface design, send us an outline.■

Editor/Publisher..... *Art Carlson*
Art Director..... *Joan Thompson*
Production Assistant..... *Judie Overbeek*
Technical Editor..... *Lance Rose*
Technical Editor..... *Phil Wells*

The Computer Hacker® is published 12 times a year. Annual subscription is $24 in the U.S., $30 in Canada, and $39 in other countries.

Entire contents copyright © 1983 by The Computer Hacker.

Postmaster: Send address changes to: The Computer Hacker, P.O. Box 1697, Kalispell, MT 59903-1697.

Address all editorial, advertising and subscription inquiries to: The Computer Hacker, P.O. Box 1697, Kalispell, MT 59903-1697.

FILE TRANSFER PROGRAMS FOR CP/M

by Lance Rose, Technical Editor

Bypassing Incompatible Diskette Formats

Although CP/M has by now established itself as the "standard"8-bit microprocessor operating system, this doesn't mean by a long shot that programs are easily transferrable from one computer running CP/M to another. The CP/M operating system includes within it a section known as the Basic I/O System or "BIOS" for short. This part of the system is very machine-dependent and is what adapts the other (standars) parts of CP/M to the particular hardware it is being used on. These machine-dependent parameters include things like the port addresses for keyboard and printer, the disk controller type and the size and format of the particular disk system involved. For this reason a version of CP/M with a BIOS written for, say, the North Star Horizon would be of no use on a Morrow Micro Decision or other system with even a slightly different hardware configuration.

Since it is often necessary and desirable to move a program from one system to another, a way must be found to overcome the differences in all these versions of what is the "standard" operating system. One way that allows a speedy transfer is to simply have a version of CP/M with a custom BIOS written for multiple disk systems. This would be used in a machine with two or more disk controllers operating at the same time. Each controller could have a drive connected to it and assigned a logical drive name, for example an 8-inch single density floppy might be Drive A, a North Star minidisk drive might be Drive B, a TRS-80 CP/M format disk drive might be Drive C, and so on. I think you can see right away that this isn't a very practical setup. One would have to have all these types of disk controller active in the same machine at the same time (a near impossibility considering the various schemes used for addressing disk controllers). Also, it's questionable whether any machine has enough slots for all the different types of hardware required, and the BIOS would be quite long and involved.

Another way of moving files around which, while slower, is at least more practical, is to simply transmit the file from one computer to another via some sort of interface. This method has the advantage of not requiring the same disk system, in fact it doesn't require the same anything except the interface convention (i.e. RS-232, Centronics, Etc.) and that both machines have CP/M running on them. The connection between the two systems may be a simple interface cable or it may have a pair of modems and a telephone line between them, thus allowing remote transfer of files. The modem method is, of course, much slower since it is no problem at all to send files from one machine to another on an RS-232 cable at 9600 baud (some 32 times faster than most modems are capable of!) Still, it is sometimes impossible to place the computers physically side-by-side and modems may have to be used.

Transfer Conventions

With transfers between machines using a hardwired cable, it isn't usually necessary to add a checksum to insure data integrity (it doesn't hurt, however), since interface reliability in the absence of a phone line should be quite good. However the case of transfer via modems is much different. Telephone line quality can vary from good to atrocious (more often the latter than the former) and some means must be adopted to make sure that what arrives at the receiving end is the same information that started out at the sending end. This is where the checksum comes in. For each block of data sent the sum of the bytes transmitted is calculated and at the end of transmission of the block, this "checksum" is also transmitted. At the receiving end the computer is adding up the values of the bytes received one by one as they come in. When the final data byte has been received, the checksum is transmitted and examined by the receiving machine. It then compares the checksum received with the one it has been calculating and, if these are the same, it is assumed that the block has been transferred correctly.

While an in-depth explanation of error-detecting and error-correcting codes is not appropriate here, suffice it to say that the probability that there will be two errors in transmission which cause equal and opposite results is so minute as to not be worth worrying about. In a case where something like national security is involved, more elaborate error-detecting and correcting codes are available to cover this possibility but for our use they are not needed. In addition they would slow down what is already a painfully slow method of moving data between computers.

Choices of Data Format

There are basically two ways that data can be represented during transmission from one computer to another. The first of these is simple ASCII coding. This works fine for text files but runs into a bit of a snag for machine-language or executable programs. ASCII is defined as a 7-bit code with a parity bit added as the 8th bit. However, binary files may have any combination of bit patterns making up the byte and can't afford to waste the 8th bit as a parity bit. In the case of some existing file transfer programs, a binary file must first be converted into a form that is representable by ASCII characters (a HEX file) before transmission. It is then transmitted and reconverted into a binary file at the receiving end. While this works, it forces the transmission of two bytes of data for each byte of binary information that must be transmitted, thus in effect cutting the transmission speed in half. Using this method would limit the effective data transfer rate on a modem to around 15 bytes/second. As I mentioned above, even 30 bytes/second is irritatingly slow especially when transferring long programs, not to mention

the expense if this is happening long distance. The alternative to this, which I am using here, is to make sure the serial port used is configured for 8-bit words and no parity bit. Most any serial port can be configured this way with a little snooping in the user's manual. In fact I have found that most hardware manufacturers use this is the default configuration for their serial data ports. With this accomplished, a binary byte can be transmitted as is, and any error-detecting can be left to the checksum rather than the parity bit.

The Programs

The programs presented here are designed to work with each other in moving files between CP/M machines. The basic method of transfer is to have the receiving machine in control of the situation. The transmitting machine waits until the receiving machine is ready before sending anything. This allows for the case where the receiving machine may have a slow disk system and a large file is being transferred that can't be buffered in memory in its entirety. The receiving machine must pause to dump its buffer to the disk, and during this period the transmitting machine must wait to insure that it will not be transmitting when the receiving machine is busy with its disk work. Upon dumping the buffer, the receiving machine can signal that it is ready to begin accepting data again and the transmitting machine can start sending at that time.

An additional feature present is a certain amount of error-correcting. The term "correcting" is a bit of a misnomer because it is accomplished here by simply retransmitting a garbled record until it is received correctly. There is a two-way communication between the machines (full duplex) so that they can decide when a record has been correctly received. The number of retries for a badly-transmitted record is 4 here but can be altered to any other value to suit the user's purpose. Aborting a transfer is also possible since the program polls the console device using a BDOS call, and if the operator types a control-C, the transfer is terminated and a message so stating is printed on the screen. As each record is transmitted and received, a message is printed on the screen so that the operator can monitor the process. If a record must be transmitted, the word "again" is appended to the message. When the entire file has been finally transferred, the message "Transfer Complete" is printed and the program reboots.

Procedure for Transferring a File

The procedure to follow in using these programs, once they have been entered into the machine, is quite simple. First of all, the user must identify the port numbers and input and output flag bits for the serial port concerned. This information is almost always available as part of the user documentation for the system. These values must then be inserted into the source listing for the programs and the programs assembled.

If the connection is a hardwired cable it must connect the serial interface on the first machine to the serial interface on the second. In most cases a cable may be needed that reverses pins 2 and 3 of the DB25 connector on one end since it will probably be the case that both machines are wired to connect directly to printers and will use pin #2 for received data.

If the connection is being made via a phone line and modems, each machine should have a cable suitable for connecting it to a modem. Most computers are wired as DCE (Data Communications Equipment, i.e. they emulate a modem) and will probably need a crossover cable to connect to an actual modem, but this is not universal so consult your user manual on this.

Since the receiving program is in control of the process, the transmitting computer should begin first with the operator typing 'TX (drive:)filename(.filetype)' where the items in parentheses, the drive and filetype, are optional. The quotes are not entered. If the filename is omitted, an error message results and the program reboots. After waiting a few seconds for the computer to open the file and load the buffer, the receiving operator types 'RX (drive:)filename(.filetype)'. His computer then erases any old file by that name, opens a new file and signals the transmitting computer to begin. At this point the computers may be left alone until the process is complete barring any unrecoverable error conditions.

In practice, even though the receiving computer is calling the shots, I have found that it doesn't seem to matter who actually types his carriage return first, the sending or the receiving party. The handshake link is established satisfactorily either way and the transfer proceeds normally. So don't worry too much about counting to five or whatever before hitting return.

Multiple File Transfers

In order to keep the complexity of the programs down, it was necessary to limit the transfer to a single file for each execution of the program. This is not really a problem in the case of long files since one would want to check on the progress of the transfer periodically and re-executing the program for the next long file wouldn't be a burden. In the case of a large number of small files, I have found that the best procedure to accomplish this is the SUBMIT utility of CP/M. Making up a submit file such as:

```
TX PROG1.TY1
TX PROG2.TY2
TX PROG3.TY3
TX PROG4.TY4
```

and calling it SEND.SUB allows the whole thing to operate by typing in SUBMIT SEND. The SUBMIT program then executes each line in turn until all files have been transferred. The receiving end computer must of course have a similar file but with the letters RX in place of TX on each line. Our procedure here is to first send the submit file with a manual command, then have the receiving end operator edit it and replace the TX's with RX's. This helps insure that the order of the programs being transferred will be the same on both ends. The receiving computer operator can then type SUBMIT RECEIVE (assuming he has named the file RECEIVE.SUB). We have used this procedure to transfer series of files that take an hour or more via modems and,

except for checking the machine every ten minutes or so, both operators can work on something else during the transfer process.

Summing Up

These programs offer the hacker a good way to move files between CP/M machines, even with toatlly different disk systems. The only requirement is that each have a serial port. Transfer can be made with either a direct interface cable or a pair of modems. (Why is it I always want to make the plural of modems moda?) The programs are very tolerant of timing and contain the facility for retransmitting records containing transmission errors. We have been using them in this area for quite some time now and found them to be very satisfactory for our purposes. The listings are included here for those who want to key them in. If you would rather avoid the effort and errors involved with keyboarding the data, send $15 to The Computer Hacker for an 8-inch single-density floppy disk containing a CP/M copy of the source files.

Listing 1
Program to transmit a CP/M file

```
;       Program to transmit a CP/M file through a serial port
;       Accesses port directly, bypassing the BDOS
;       Assumes an 8-bit word length and no parity
;
;       Version of 9/2/83
;
;
BOOT    EQU     0000H           ;CP/M reboot address
BDOS    EQU     0005H           ;CP/M BDOS entry point
TFCB    EQU     005CH           ;FCB for file to be transmitted
SIOSTA  EQU     07H             ;SIO status port
SIODAT  EQU     06H             ;SIO data port
IFLAG   EQU     02H             ;Input flag for serial port
OFLAG   EQU     01H             ;Output flag for serial port
BUFREC  EQU     128             ;Buffer size (CP/M records)
;
;
        ORG     100H
;
;       If required, place serial port initialization code here
;
TX:     LDA     TFCB+1
        CPI     ' '             ;Check for filename
        JNZ     OPEN
        LXI     D,FNMER         ;Print error message and reboot
ABORT:  MVI     C,9
        CALL    BDOS
        MVI     E,04H
        CALL    XMTBYT          ;Send EOT character
        JMP     BOOT            ;Return to CP/M
OPEN:   LXI     D,TFCB
        MVI     C,15
        CALL    BDOS            ;Oper file
        INR     A
        JNZ     FOUND           ;File present
        LXI     D,FNFER         ;Point at error message
        JMP     ABORT           ;Print and reboot
FOUND:  XRA     A
        STA     TFCB+32         ;Set next record byte to zero
        MOV     C,A             ;Set record count in buffer to zero
READY:  PUSH    B
        MVI     C,11
        CALL    BDOS            ;Get console status
        POP     B
        ORA     A
        JZ      READY1          ;No key pressed
        PUSH    B
        MVI     C,1
        CALL    BDOS
        POP     B
        CPI     03H
        JNZ     READY1
ENDXMT: LXI     D,EOTMSG        ;Print EOT message and abort
        JMP     ABORT
READY1: IN      SIOSTA          ;Read serial status port
        ANI     IFLAG
        JZ      READY           ;Wait for character to be received
        IN      SIODAT          ;Get received character
        ANI     7FH             ;Mask off bit 7 for ASCII codes
        CPI     04H
        JZ      ENDXMT          ;End transmission
        CPI     01H
        JZ      XMTREC          ;Transmit next record
        CPI     02H
        JNZ     READY           ;Ignore other characters
        LHLD    DATPTR
        LXI     D,-0080H
        DAD     D               ;Point at last record
        SHLD    DATPTR
        INR     C               ;Go back one record
XMTREC: DCR     A
        STA     RPTFLG          ;Save repeat flag
        CZ      COUNT           ;Increase record message if new record
XMTRC1: DCR     C               ;Decrease record count
        JP      XMTRC2          ;More in buffer
        LDA     EOFFLG
        ORA     A
        JZ      READ1           ;More in file
        LXI     D,EOFMSG                                  continued
```

Listing 1, continued

```
        MVI     C,9
        CALL    BDOS            ;Print EOF message
        MVI     E,03H
        CALL    XMTBYT          ;Send EOF byte
        JMP     BOOT
READFL: LXI     H,DATBUF
        SHLD    DATPTR          ;Point at beginning of data buffer
        XCHG
        INR     C               ;Initialize record count to zero
READ:   PUSH    B
        PUSH    D
        MVI     C,26
        CALL    BDOS            ;Set DMA address
        LXI     D,TFCB
        MVI     C,20
        CALL    BDOS            ;Read next record
        POP     D
        POP     B
        ORA     A
        JNZ     NOMORE          ;EOF detected
        LXI     H,0080H
        DAD     D               ;Point at next record in buffer
        XCHG
        INR     C               ;Increase record count
        MVI     A,BUFREC
        CMP     C
        JNZ     READ            ;Stop at end of extent
        JMP     XMTRC1
NOMORE: MVI     A,1
        STA     EOFFLG          ;Set EOF flag
        JMP     XMTRC1
XMTRC2: LDA     RPTFLG
        INR     A
        MOV     E,A
        CALL    XMTBYT          ;Send response to request for record
        LHLD    DATPTR          ;Point at next record to transmit
        MVI     B,80H           ;Character count for record
        MVI     D,0             ;Checksum for record
XMTRC3: MOV     A,M             ;Get next byte to send
        INX     H               ;Update pointer
        MOV     E,A             ;Save in E
        ADD     D
        MOV     D,A             ;Update checksum
        DCR     B
        JNZ     XMTRC3          ;More in record
        MOV     E,D
        CALL    XMTBYT          ;Send checksum
        PUSH    B
        MVI     C,4
        MVI     E,00H
XMTRC4: CALL    XMTBYT          ;Send 4 NULs to replace any missed bytes
        DCR     C
        JNZ     XMTRC4
        SHLD    DATPTR          ;Save pointer to next record
        LXI     D,RECMSG
        MVI     C,9
        CALL    BDOS            ;Print record message
        LDA     RPTFLG
        ORA     A
        JZ      NEXTLN          ;Skip 'again' if successful first time
        LXI     D,AGAIN
        MVI     C,9
        CALL    BDOS            ;Print 'again' if record repeated
NEXTLN: LXI     D,CRLF
        MVI     C,9
        CALL    BDOS            ;Print CRLF
        POP     B
        JMP     READY           ;Go wait for next prompt to send
COUNT:  LXI     H,RECCNT+4
COUNT1: INR     M
        MVI     A,'9'
        CMP     M               ;Over 9?
        RNC
        MVI     M,'0'
        DCX     H
        MOV     A,M
        CPI     ' '
        JNZ     COUNT1
        MVI     M,'0'
        JMP     COUNT1          ;Put 0 in message
XMTBYT: IN      SIOSTA          ;Transmit character through serial port
        ANI     OFLAG
        JZ      XMTBYT
        MOV     A,E
        OUT     SIODAT
        RET
;
FNMER:  DB      'File name missing',0DH,0AH,'$'
FNFER:  DB      'File not found',0DH,0AH,'$'
EOFMSG: DB      'Transfer complete',0DH,0AH,07H,'$'
EOTMSG: DB      'Transfer terminated',0DH,0AH,07H,'$'
RECMSG: DB      'Record #'
RECCNT: DB      '    0'         ;Record # transmitted
        DB      ' transmitted$'
AGAIN:  DB      ' again$'       ;Repeated record
CRLF:   DB      0DH,0AH,'$'     ;CRLF sequence
DATPTR: DW      DATBUF          ;Pointer to next data byte to send
EOFFLG: DB      0               ;Flag for EOF read
RPTFLG: DB      0               ;Flag for repeated record
DATBUF  EQU     $               ;Data buffer
;
        END
```

Listing 2
Program to receive a CP/M file

```
;       Program to receive a CP/M file through a serial port
;       Accesses port directly, bypassing the BDOS
;       Assumes an 8-bit word length and no parity
;
;       Version of 9/17/83
;
;
BOOT    EQU     0000H           ;CP/M reboot address
BDOS    EQU     0005H           ;CP/M BDOS entry point
TFCB    EQU     005CH           ;FCB for file to be transmitted
SIOSTA  EQU     07H             ;SIO status port
SIODAT  EQU     06H             ;SIO data port
IFLAC   EQU     02H             ;Input flag for serial port
OFLAC   EQU     01H             ;Output flag for serial port
BUFREC  EQU     128             ;Buffer size (CP/M records)
RETRY   EQU     4               ;Number of retries before quitting
;
;
        ORG     100H                                      continued
```

Listing 2 continued

```
;
;       If required, place serial port initialization code here
;
RX:     LDA     TFCB+1
        CPI     ' '             ;Check for filename
        JNZ     OPEN
        LXI     D,FNMER         ;Print error message and reboot
ABORT:  MVI     C,9
        CALL    BDOS
        MVI     E,04H
        CALL    XMTBYT          ;Send EOT character
        JMP     BOOT            ;Return to CP/M
OPEN:   LXI     D,TFCB
        MVI     C,19
        CALL    BDOS            ;Delete old file if present
        LXI     D,TFCB
        MVI     C,22
        CALL    BDOS            ;Make new file
        INR     A
        LXI     D,DEFER         ;Point at error message
        JZ      ABORT           ;Print and reboot
START:  MVI     C,30            ;Send some NULs first
        MVI     E,00H
NULS:   CALL    XMTBYT
        DCR     C
        JNZ     NULS
        IN      SIODAT          ;Clear serial data port
READY:  LDA     RPTCTR
        CPI     RETRY
        MVI     E,01H
        JZ      READY1          ;First time for this record
        INR     E               ;Change request character to 02H
        ORA     A               ;Test repeat counter
        JNZ     READY1          ;Try again if not zero
ENDXMT: LXI     D,EOTMSG        ;Print EOT message and exit
        JMP     ABORT
READY1: CALL    XMTBYT          ;Send prompting byte
READY2: MVI     C,11
        CALL    BDOS            ;Get console status
        ORA     A
        JZ      READY3          ;No key pressed
        MVI     C,1
        CALL    BDOS
        CPI     03H
        JZ      ENDXMT          ;Terminate if control-C pressed
READY3: IN      SIOSTA          ;Read serial status port
        ANI     IFLAG
        JZ      READY2          ;Wait for character to be received
        IN      SIODAT          ;Get received character
        ANI     7FH             ;Mask off bit 7 for ASCII codes
        CPI     03H
        JZ      CLOSE           ;Flush buffer and close file
        CPI     04H
        JZ      ENDXMT          ;End transmission
        CPI     01H
        JZ      RCVREC          ;Receive next record
        CPI     02H
        JNZ     READY2          ;Ignore other characters
RCVREC: MVI     B,66H           ;Byte count for record
        MVI     D,0             ;Initialize checksum
        LHLD    DATPTR          ;Prepare to store data
RCVRC1: CALL    RCVBYT          ;Get a byte
        MOV     M,A             ;Store in data buffer
        INX     H
        ADD     D
        MOV     D,A             ;Update checksum
        DCR     B
        JNZ     RCVRC1          ;Continue for 126 bytes
        CALL    RCVBYT          ;Get checksum
        CMP     D
        PUSH    PSW             ;Save status
        JNZ     RCVRC2          ;Bad read
        SHLD    DATPTR          ;Save new record pointer
RCVRC2: LXI     D,RECMSG
        MVI     C,9
        CALL    BDOS            ;Print received record message
        LDA     RPTCTR
        CPI     RETRY
        JZ      RCVRC3          ;First try
        LXI     D,AGAIN
        MVI     C,9
        CALL    BDOS            ;Print 'again'
RCVRC3: LXI     D,CRLF
```

continued

```
        MVI     C,9
        CALL    BDOS            ;Print CRLF
        LXI     H,RPTCTR
        DCR     M               ;Decrement repeat counter
        POP     PSW
        JNZ     START           ;Unsuccessful read
        MVI     A,RETRY
        STA     RPTCTR          ;Reset counter for next record
        LXI     H,RECCNT+4
COUNT:  INR     M
        MVI     A,'9'
        CMP     M               ;Over 9?
        JNC     BUFCHK
        MVI     M,'0'
        DCX     H
        MOV     A,M
        CPI     ' '
        JNZ     COUNT
        MVI     M,'0'
        JMP     COUNT           ;Put 0 in message
BUFCHK: LHLD    DATPTR
        LXI     D,-(DATBUF+BUFREC*128)
        DAD     D
        CC      FLUSH           ;Flush buffer if full
        JMP     READY           ;Go look for next record
FLUSH:  LXI     D,DATBUF        ;Start at beginning of buffer
FLUSH1: LHLD    DATPTR
        MOV     A,D             ;Compare to see if empty
        CMP     H
        JNZ     FLUSH2          ;More to go
        MOV     A,E
        CMP     L
        JNZ     FLUSH2
        LXI     H,DATBUF
        SHLD    DATPTR          ;Reset data pointer
        RET
FLUSH2: PUSH    D
        MVI     C,26
        CALL    BDOS            ;Set DMA address
        LXI     D,TFCB
        MVI     C,21
        CALL    BDOS            ;Write record
        POP     D
        ORA     A
        JZ      FLUSH3          ;Good write
        LXI     D,DEFER         ;Disk error
        JMP     ABORT
FLUSH3: LXI     H,0080H
        DAD     D               ;Point to next record
        XCHG
        JMP     FLUSH1          ;Go write it if present
CLOSE:  CALL    FLUSH           ;Flush buffer of data
        LXI     D,TFCB
        MVI     C,16
        CALL    BDOS            ;Close file
        LXI     D,EOFMSG
        MVI     C,9
        CALL    BDOS            ;Print EOF message
        JMP     BOOT            ;Reboot
RCVBYT: IN      SIOSTA          ;Receive byte from serial port
        ANI     IFLAG
        JZ      RCVBYT
        IN      SIODAT
        RET
XMTBYT: IN      SIOSTA          ;Transmit byte to serial port
        ANI     OFLAG
        JZ      XMTBYT
        MOV     A,E
        OUT     SIODAT
        RET
;
FNMER:  DB      'File name missing',0DH,0AH,'$'
DEFER:  DB      'Disk or directory full',0DH,0AH,'$'
EOFMSG: DB      'Transfer complete',0DH,0AH,07H,'$'
EOTMSG: DB      'Transfer terminated',0DH,0AH,07H,'$'
RECMSG: DB      'Record #'
RECCNT: DB      '    1'         ;Record # received
        DB      ' received$'
AGAIN:  DB      ' again$'       ;Repeated record
CRLF:   DB      0DH,0AH,'$'     ;CRLF sequence
DATPTR: DW      DATBUF          ;Pointer to next storage location in buffer
RPTCTR: DB      RETRY           ;Counter for repeated record
DATBUF  EQU     $               ;Data buffer
;
        END
```

■

A Challenge to FORTH Advocates...

Our readers are involved with interfacing and control, and are interested in hearing more about FORTH. Here is your chance to convince them of the advantages of FORTH.

Submit your outline or articles (SASE appreciated) for prompt consideration. Author's guide available. Write to:

The Computer Hacker
P.O. Box 1697, Kalispell, MT 59903-1697

THE RS-232-C SERIAL INTERFACE

Part Two

by Phil Wells—Technical Editor

Introduction

The first part of this article discussed the electrical, mechanical and functional specifications of the EIA RS-232-C serial interface standard. Part two will briefly discuss the "standard configurations" defined in RS-232-C, then describe some real-world configurations and present several suggested "hacker" standards.

Standard Configurations

The RS-232-C standard defines 13 "Standard Interfaces," called "Interface Type" A through M, with a fourteenth catagory called Interface Type Z for specials defined by the manufacturer. I think every "RS-232-C compatible" interface I've ever seen in microcomputer equipment has been "Type Z," including those found on typical low-cost modems.

The standard interface types are defined in terms of which interchange circuits are implemented. All standard configurations include a number of circuits required for that type, plus possibly some circuits required for switched service, some required for synchronous service, and some optional circuits.

Keep in mind that the standard defines a serial interface between a computer or terminal (Data Terminal Equipment—DTE) and a MODEM or Data Set (Data Communications Equipment—DCE). RS-232-C was not intended to define an interface between a computer and printer, or directly between two computers.

Some often-misused terms apply to the interface type descriptions:

Simplex: One-way-only transmission. Not reversible.

Half Duplex: Two-way transmission, but only one way at a time.

Full-Duplex: Two-way simultaneous transmission. Often mistaken for "Echo" or "Echoplex". An echo is when the characters you type on your keyboard are not immediately displayed on your screen; instead they are sent to the remote computer which echoes or returns them to your terminal which then displays them. This is a simple but very good means of error checking for character-at-a-time transmissions. If your terminal sends your typed characters directly to your screen and the remote system is echoing, you will see a double of every character you type. If your terminal software requires a remote echo but the remote computer is not set up to provide an echo, you will see nothing on your screen when you type; in this case your modem may provide a local echo if you switch it to Half Duplex.

Switched Service: Additional control circuits are required if the link includes switched communication circuits. Generally, this means that if you have a dedicated (non-switched) set of wires connecting the two Data Sets, you do not need complete handshaking between the DTE and DCE. However, if you are connected to the PSTN (Public Switched Telephone Network) or to some other arrangement where the data sets might not always be connected, you are required to include the additional control interchange circuits.

The simplest standard type requires four wires (figure 1): Signal Ground, Transmitted Data, Clear To Send, and Data Set Ready. Data Terminal Ready and Ring Indicator are required for switched service. This configuration, Type A, is a transmit-only Simplex (meaning one-way only) type interface.

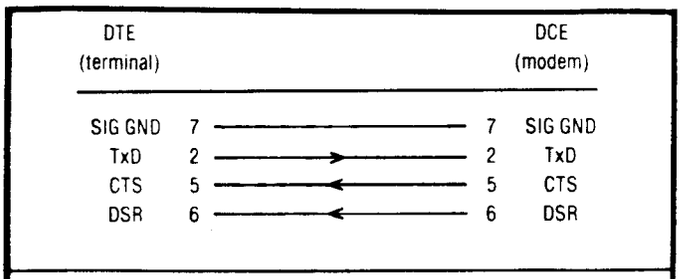


Figure 1: Standard interface type A. Simplex transmit-only.

The handshaking here is strictly one-way. Before transmitting, the DTE must check for an "on" (high) level on the CTS and DSR lines. DSR "on" means the DCE is connected to a communication channel, is not in test, talk or dial modes, and has completed any answer tone and timing functions. CTS "on" means the data set (DCE) is ready to transmit data to the communication channel.

Interface type B (figure 2) is the same as type A with the addition of the Request to Send line, by which the DTE can tell the DCE that it wants to transmit. This allows the DCE to disconnect from the channel between transmissions. Ring Indicator is required for switched service.

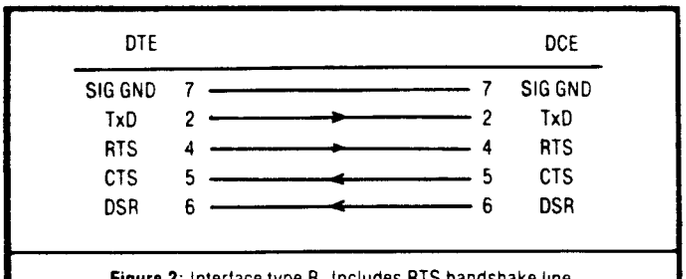


Figure 2: Interface type B. Includes RTS handshake line.

The other four-wire interface is Type C, a Simplex receive-only interface using Signal Ground, Received Data, Data Set Ready, and Received Line Signal Detector (Data Carrier

Detector). See figure 3. There is no handshaking involved here, except that if either DSR or DCD is false, the DTE knows the DCE will not transmit data.

Interface types A-E define primary channel only interfaces; the rest include a secondary channel.

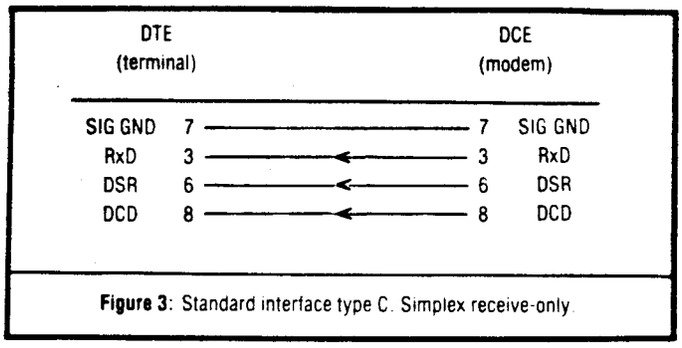


Figure 3: Standard interface type C. Simplex receive-only.

The simplest full-duplex (two-way simultaneously) configuration is interface type E (figure 4). This requires six wires: Signal Ground, Transmitted Data, Received Data, Clear to Send, Data Set Ready, and Received Line Signal Detector.

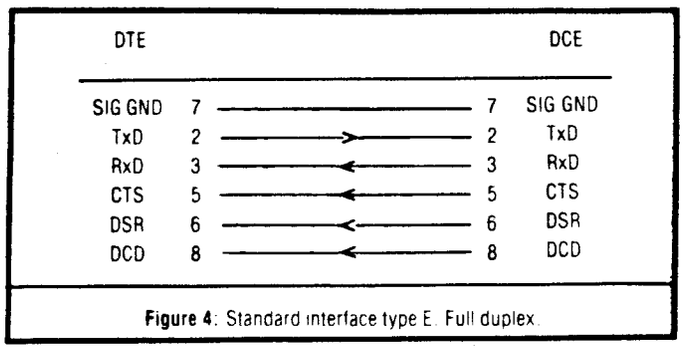


Figure 4: Standard interface type E. Full duplex.

The remaining standard configurations are much more complex than needed for most simple tasks. In the real world of low-cost microcomputer equipment we seldom see any of these "standard" types.

The Real (Micro) World.

The remaining discussion concerns communication between a computer and printer or between two computers, etc.; not between a computer and a modem.

The purpose of the control lines is to ensure that nobody tries to send data unless someone is ready to receive. If the control lines are not used, things go all right until the receiving device's input buffer overflows. Figure 5 shows a very common three-wire interface between a computer and a printer. Note that the leads between pins 2 and 3 "cross over" since both the computer and printer are set up internally as DTE. THis works if the printer can print faster than the computer sends data. For example, a Tally 1805 printer printing at 200 characters per second can stay ahead of a computer sending at 1200 BAUD (about 120 characters per second). But if an 80-character per second printer (or a higher BAUD rate) is used, large chunks of text will not be printed; when the printer's input buffer overflows, data is simply lost. Some printers will sound a warning buzzer, turn on an indicator light and stop printing when an input buffer overflow occurs.

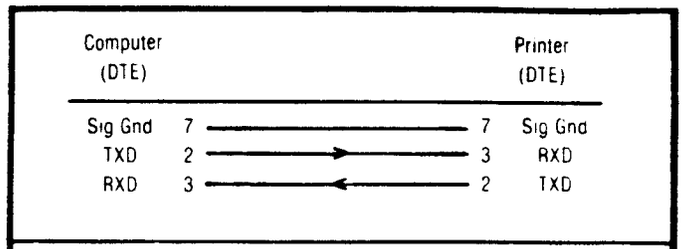


Figure 5: Simple 3-wire printer cable, where the computer is DTE.

Notice in Table 1 there are two circuits for the DTE (printer) to send control signals to the DCE: RTS and DTR. The Request to Send line, when high, tells the DCE that the DTE wants to send data; the DCE usually responds with a

Pin	RS-232-C	CCITT	Mneumonic	Description
1	AA	101	GND	Protective Ground
2	BA	103	TxD	Transmitted Data
3	BB	104	RxD	Received Data
4	CA	105	RTS	Request to Send
5	CB	106	CTS	Clear to Send
6	CC	107	DSR	Data Set Ready
7	AB	102	GND	Signal Ground
8	CF	109	DCD	Rcvd Line Signal Det.
19	SCA	120	SRTS	Secondary RTS
20	CD	108.2	DTR	Data Terminal Ready
22	CE	125	RI	Ring Indicator

Table 1

Clear to Send. When the DTE is ready to receive data (printer's buffer is empty, paper and ribbon are O.K.) it raises the Data Terminal Ready line high. The simplest and most common way to add a hardware handshake line to the three wire interface shown in figure 5 is to add a DTR/DSR line, as in figure 6.

Notice also that the DCE uses five control lines to signal conditions to the DTE (primary channel only). These are Clear to Send, Data Set Ready, Data Carrier Detect, Signal Quality Detector, and Ring Indicator. The last three only make sense in the context of a modem, but you'll find that many RS-232-C DTE ports will not send unless CTS, DSR and DCD are all high (which is actually what the standard says).

Given these sets of control lines, we find some common configurations which are different designer's attempts to find a reasonably cost-effective simplification of the formal standard.

We find that the Data Terminal Ready/Data Set Ready and the Request To Send/Clear To Send pairs are the most often implemented lines, with Data Carrier Detect appearing less

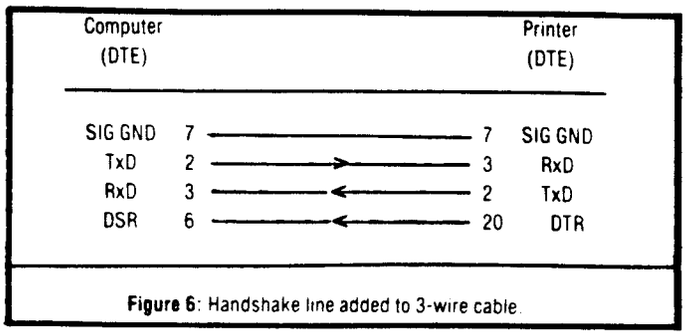


Figure 6: Handshake line added to 3-wire cable.

frequently. The remaining lines are very rare on low-cost (and some not-so-low cost) equipment.

When the DTR/DSR pair is used, the procedure is for a DTE to set DTR high when it is prepared to listen, and to check that DSR (from the DCE) is high before talking.

When the RTS/CTS pair is used, the procedure is for a DTE to set RTS high when it wants to send and to make sure the DCE has responded with CTS high before sending.

The Data Carrier Detect (Data Carrier Detected) control line often causes complications when a DTE computer is connected to a DTE printer, since there is no "complimentary" line as there is with the DTR/DSR and RTS/CTS pairs. Computer or terminal serial ports often include a DCD input because most MODEMs use a DCD output to signal that they are receiving a carrier — essential to communication between MODEMs. Again, when you want to connect two computers, both DTE, using a "MODEM eliminator" or "null MODEM" cable as shown in figure 6.

Several printers use the Secondary Request to Send (SCA — also called Reverse Channel Request to Send) line to signal the host computer that the printer wants to transmit a message — usually status or an error message — back to the computer. Although SCA is officially pin 19, at least one device calls pin 11 SCA (pin 11 is officially unassigned).

NOTATION	INTERCHANGE VOLTAGE	
Binary State	-3 to -15	+ 3 to + 15
	1	0
Signal Condition	Mark	Space
Control Function	Off	On

Table 2

Some Real Examples

Let's start with an example of what passes for a "complete" set of leads in the micro-world. Figure 7 shows the signal leads implemented in the IBM Asynchronous Communications Adapter for the IBM PC and XT models. This is about as complete as you are likely to find in personal computers. Machines designed for use in master-slave multi-station systems will have more control leads, but that is a different world. This is an "RS-232-C-like" interface permanently configured as Data Terminal Equipment. Connecting this to a Hayes SmartModem 300 is simplicity itself; a straight-thru cable works fine — but watch out! IBM put a current loop interface on pins 9, 11, 18 and 25 of its DB-25P connector. You need to be sure the device you're connecting to doesn't use these pins if you use a 25-wire cable. I just used a 9-wire cable.

Note that IBM is one of the few to actually use a male connector on the DTE, as specified by RS-232-C. Now, if only they hadn't stuck on that current loop, and if only they had used a Centronics-style connector on their parallel printer port instead of a DB-25S.

It is not possible to know exactly how the control lines on this port behave just by examining the documentation accompanying the adapter card, since this card is highly software-controlled. To know which lines are used for "handshaking", which lines generate interrupts, and which lines are ignored, you need to examine both the hardware and the software. Since you rarely have access to both (on both equipments, remember), there can be a lot of cut-and-try in the cabling.

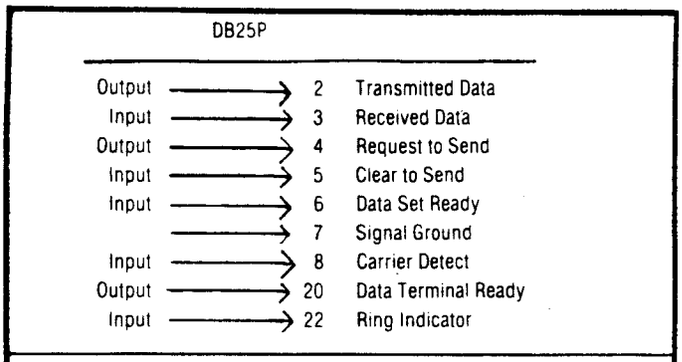

DB25P
Output → 2 Transmitted Data
Input → 3 Received Data
Output → 4 Request to Send
Input → 5 Clear to Send
Input → 6 Data Set Ready
→ 7 Signal Ground
Input → 8 Carrier Detect
Output → 20 Data Terminal Ready
Input → 22 Ring Indicator

Figure 7: IBM PC and XT Asynchronous Communications Adapter — "RS-232-C-like" interface.

Figure 8 shows the cable recommended by Apple to connect a Qume Sprint 5 or DEC LA120 printer to an Apple III computer. This illustrates several common problems. The Apple III's built-in serial port is hard-wired as DTE; so is the printer. The "MODEM eliminator" or "null MODEM" cable crosses some leads so that each device appears to the other as a DCE. Note that DTR and DSR are paired, but crossed over. Also note that the DCD inputs are driven by RTS, and that each device's CTS input is driven by its own RTS output.

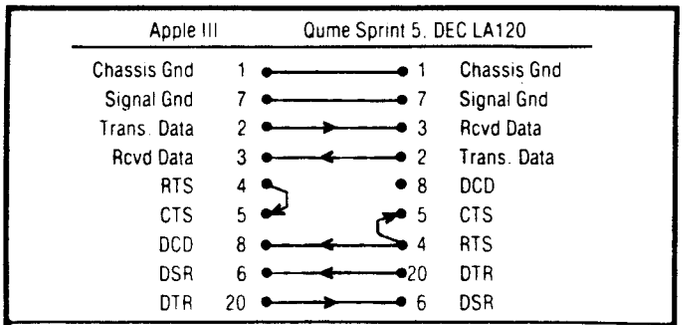

Apple III
Qume Sprint 5, DEC LA120
Chassis Gnd 1 → 1 Chassis Gnd
Signal Gnd 7 → 7 Signal Gnd
Trans. Data 2 → 3 Rcvd Data
Rcvd Data 3 → 2 Trans. Data
RTS 4 → 8 DCD
CTS 5 → 5 CTS
DCD 8 → 4 RTS
DSR 6 → 20 DTR
DTR 20 → 6 DSR

Figure 8: "Modem eliminator" cable between Apple III and Qume Sprint 5, where both devices are DTE.

Figure 9 shows Apple's recommended cable for connecting an Apple III to the IDS 440, 445, 460 and 560 line of printers. These printers do not provide software handshaking. Their serial interfaces are receive-only. They are also unusual in

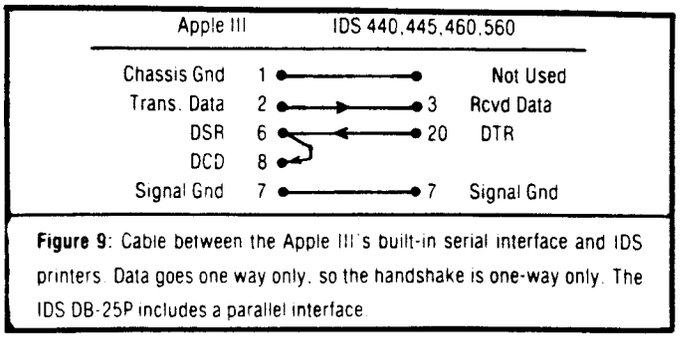


Figure 9: Cable between the Apple III's built-in serial interface and IDS printers. Data goes one way only, so the handshake is one-way only. The IDS DB-25P includes a parallel interface.

that the circuit-board mounted connector is a male DB-25, and contains a parallel interface as well as the "RS-232-C" pins. These printers signal an input buffer-full condition with the Data Terminal Ready line; this is the most commonly used line for this purpose.

Figure 10 shows an Apple III to Okidata 82A printer connection. The Okidata printer manual calls pin 11 "SCA", or "BUFFER BUSY/FULL". This pin is offically unassigned in the RS-232-C standard, which calls pin 19 SCA or "Secondary Request to Send". The excellent Mannesmann-Tally 1805 printer also provides a READY/BUSY signal on pin 11, but covers the spec by providing the same signal on pin 19. Notice that the Okidata printer is using its own DTR output to drive its DSR input. This satisfies the printer's requirement for an external pull-up on its DSR line.

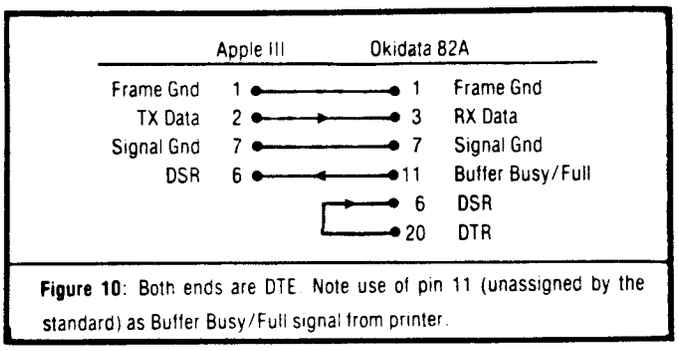


Figure 10: Both ends are DTE. Note use of pin 11 (unassigned by the standard) as Buffer Busy/Full signal from printer.

Figure 11 shows Okidata's suggested connecting cable between an 82A printer and a Radio Shack Model II's built-in serial port. Notice that the printer's busy signal (11) is connected to the computer's CTS (5) input, with the DTR/DSR pair crossed over. Compare this with figure 9.

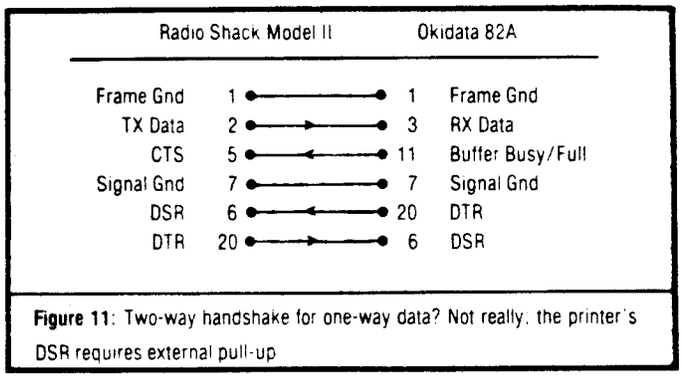

Radio Shack Model II
Okidata 82A
Frame Gnd 1 1 Frame Gnd
TX Data 2 3 RX Data
CTS 5 11 Buffer Busy/Full
Signal Gnd 7 7 Signal Gnd
DSR 6 20 DTR
DTR 20 6 DSR

Figure 11: Two-way handshake for one-way data? Not really; the printer's DSR requires external pull-up

Figure 12 is from an Apple Serial Interface card manual, purporting to show how to connect this card to a printer. Notice that the printer's control outputs are fed back to its inputs. This is because the Apple serial card contains no driver circuitry (or receivers, either). No handshaking is going on here; the jumpers are needed to satisfy the printer's control input requirements. If the printer's input buffer fills, data will be lost, so the BAUD rate must be set low enough that the printer can stay ahead of the computer. Apple has replaced this older card with the Super Serial Card, probably the best example of a universal (in terms of flexibility) serial interface card I've ever seen. Far superior to the Apple III's built-in serial port.

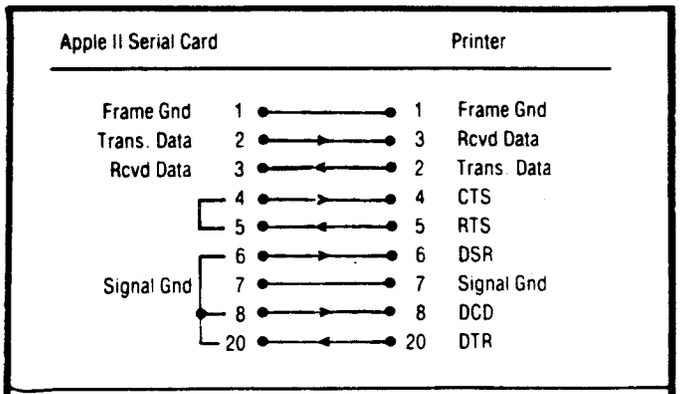

Apple II Serial Card
Printer
Frame Gnd 1 1 Frame Gnd
Trans. Data 2 3 Rcvd Data
Rcvd Data 3 2 Trans. Data
4 4 CTS
5 5 RTS
6 6 DSR
Signal Gnd 7 7 Signal Gnd
8 8 DCD
20 20 DTR

Figure 12: Real-world nearly useless interface. Notice the control lines on the Apple Serial Interface Card are simply jumped together. There is no handshaking capability. If the printer's input buffer fills, data will be lost. The BAUD rate must be set so slow that the printer can always stay ahead

Figure 13 is from the NEC 7700 series Spinwriter printers. It illustrates the implementation of a complete DTE RS-232-C interface designed to connect to a modem with additional printer control functions on the two test lines and three unassigned lines. The five additional functions are provided for use in direct-connect (no modem) situations to provide more complete control of the printer. A "standard" computer serial port would not be able to take advantage of these lines, but they would be very useful in an OEM (custom) design.

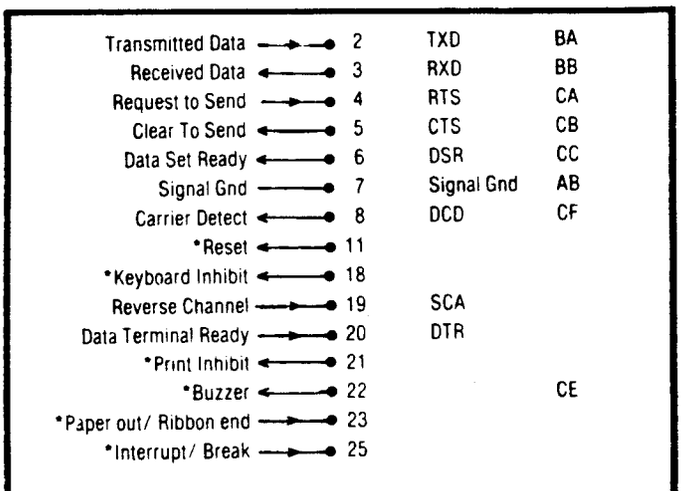

Transmitted Data 2 TXD BA
Received Data 3 RXD BB
Request to Send 4 RTS CA
Clear To Send 5 CTS CB
Data Set Ready 6 DSR CC
Signal Gnd 7 Signal Gnd AB
Carrier Detect 8 DCD CF
*Reset 11
*Keyboard Inhibit 18
Reverse Channel 19 SCA
Data Terminal Ready 20 DTR
*Print Inhibit 21
*Buzzer 22 CE
*Paper out/ Ribbon end 23
*Interrupt/ Break 25

Figure 13: NEC 7700 series Spinwriter. *Only used with keyboard version.

Figure 14, the Qume Sprint 5 serial port, illustrates what I think is an excellent idea. This looks like a fairly typical RS-232-C port, but it has a useful twist. A switch just behind the front panel of the printer but accessible without removing screws can be set to MODEM or NO MODEM. In the NO MODEM position, the CTS and DSR (and presumably DCD) inputs are ignored and either the RTS or DTR output line can be used for "hardware handshaking" in a direct connection to a computer. These lines will be turned off (negative) when the input buffer is within two characters of being full and will be turned back on only when the input

buffer drops to within ten characters of empty. If the printer is connected to a modem, the switch is set to the MODEM position and RTS, DTR, CTS, DSR and DCD become standard RS-232-C control lines.

DTE	DB25S	
Frame Gnd	1	
Trans. Data	2	Output: Software Handshake
Rcvd Data	3	Input: Data to Print
Request to Send	4	Output
Clear to Send	5	Input: Must be High
Data Set Ready	6	Input: Must be High
Signal Gnd	7	
Data Carrier Detect (Rcvd line signal detector)	8	Input: Must be High
Data Terminal Ready	20	Output: High = Ready, Low = Busy

Figure 14: Qume Sprint 5. Inputs 5,6 and 8 are ignored when the Sprint 5 is switched to "No Modem."

Recommendations for Hacker Projects

We will find three and possibly four serial interface situations in our construction projects: simplex send-only, simplex receive-only, full duplex and possibly half-duplex. Recall that half-duplex means two-way communication, but only one direction at a time. Full-duplex means two ways at the same time.

Most serial interfaces today use an integrated circuit called a UART or USART. This device provides most of the circuit functions needed to convert a microprocessor's parallel data to serial, to buffer the received and transmitted data, and to control at least a few of the interface lines (USARTs). These devices will be covered in detail in a future article. For now, be confident that one of these devices, a few support chips and a little software makes it easy for us to implement any of the three or four configurations needed. The limitation is that these chips usually provide at most three or four of the RS-232-C Control Interchange functions, so we need to decide which ones to use and keep these to a minimum.

We need signal ground, transmit data or receive data or both, and some way for a receiving device to indicate a busy eady status. For a design where we don't know what will be on the other end of the cable, we should provide more of the control lines. We can save effort by using a jumper arrangement to "configure" the port.

To avoid confusion over lead names, Table 1 gives the RS-232-C names, the common-use mnemonic names, the CCITT names, and the descriptive names of the interchange circuits we will be using. Table 2 should remind you that an ON condition on a control line is a positive voltage.

A simple receive-only design recommendation is shown in figure 15a. I'm going to assume we are making Data Terminal Equipment; if your design is for a DCE, connect the pins shown in parentheses. Data is received on pin three; the RS-232-C line receiver (inverting) meets the termination requirements. The interface signals that it is busy by putting a negative voltage on the DTR line.

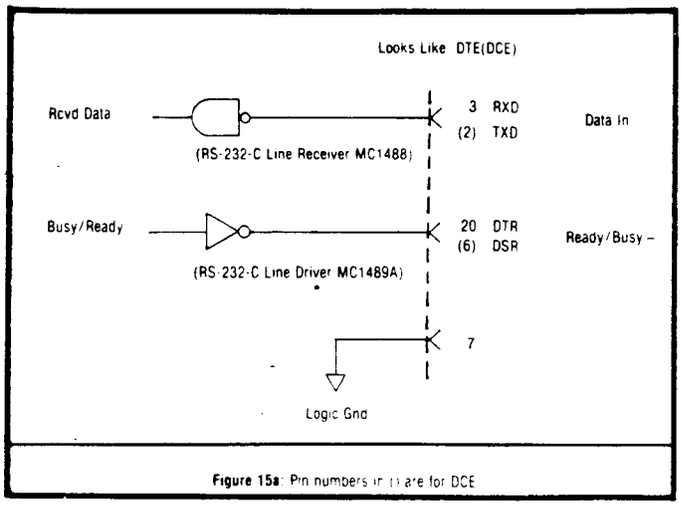


Figure 15a: Pin numbers in ( ) are for DCE

Figure 15b is a suggestion for making this receive-only design more flexible. The jumper blocks can be the dual rows of pins spaced 0.1 inch apart, available from Radio Shack and many mail order firms. These are conveniently jumpered with small, two-pin female blocks, or with wire-wrap wire. These jumper blocks allow you to configure the hardware as either DTE or DCE. The dashed lines indicate the "most standard" connections. Note that you can get away with driving more than one line (e.g., CTS, DSR and DCD) from one MC1489A, even though this might not exactly meet the RS-232-C specs.

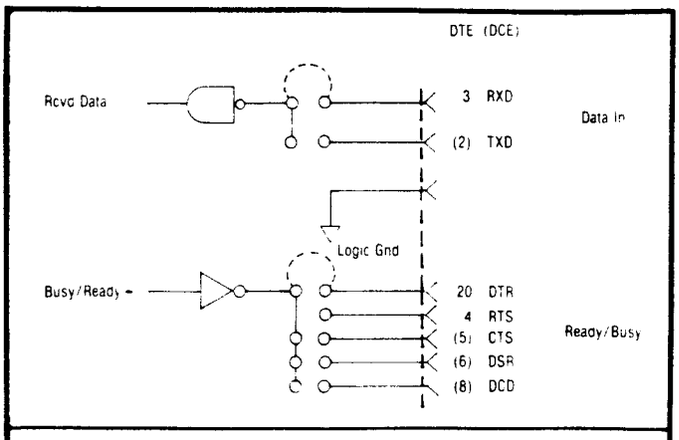


Figure 15b: Simplex receive-only with DTR Ready/Busy. Pin numbers in ( ) are for DCE

Figure 16 is a suggestion for a transmit-only interface. Eliminate the jumpering if you have a dedicated situation; this arrangement makes it possible to accidentally short multiple line driver outputs together (e.g., CTS, DSR and DCD outputs from a DCE). The DTR and SCA ready/busy-lines are provided in case this port must be configured as DCE to drive a printer which signals buffer full on pin 20 or 19. Unfortunately, a fair number of printers use the SCA (Secondary Request to Send) as a "handshake" line.

The full or half duplex (controlled by software) general-

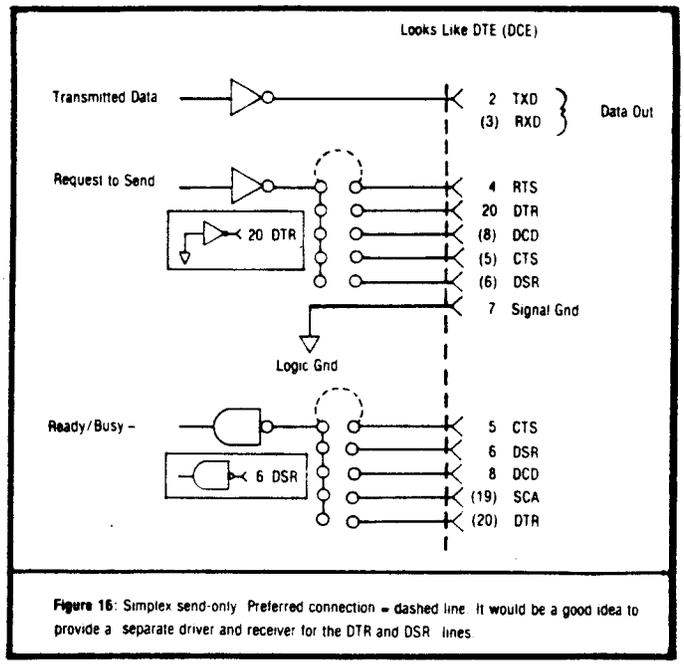


Figure 16: Simplex send-only. Preferred connection = dashed line. It would be a good idea to provide a separate driver and receiver for the DTR and DSR lines.

purpose DTE interface in figure 17 assumes you are using a USART which provides two output and four input control signals. These are inexpensive and readily available today. For most applications, one output and one input control will work; which ones you select depends on what parts are

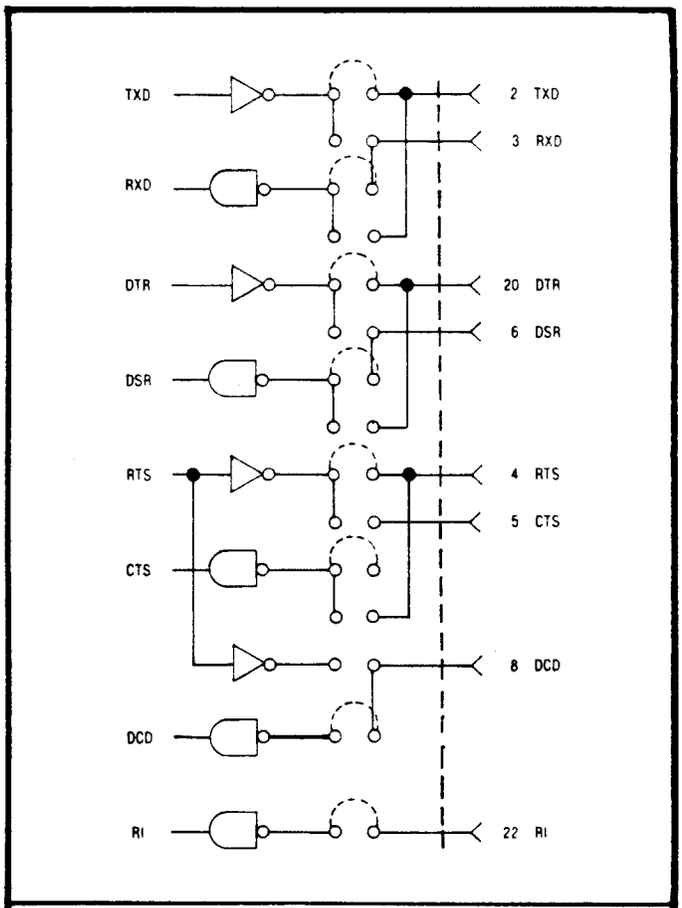


Figure 17: Full duplex general purpose DTE. can be jumpered to a DCE configuration.

available to you. The most commonly omitted input is the Ring Indicator, since this is only used with a direct-connect modem. Figure 18 is a suggested layout for use with USARTs having only two control inputs and two control outputs.

Given the variety of configurations of both DCE and DTE with which your general-purpose DTE interface may have to work, it may be simpler to forget the jumpering and build custom cables such as those shown earlier in this article.

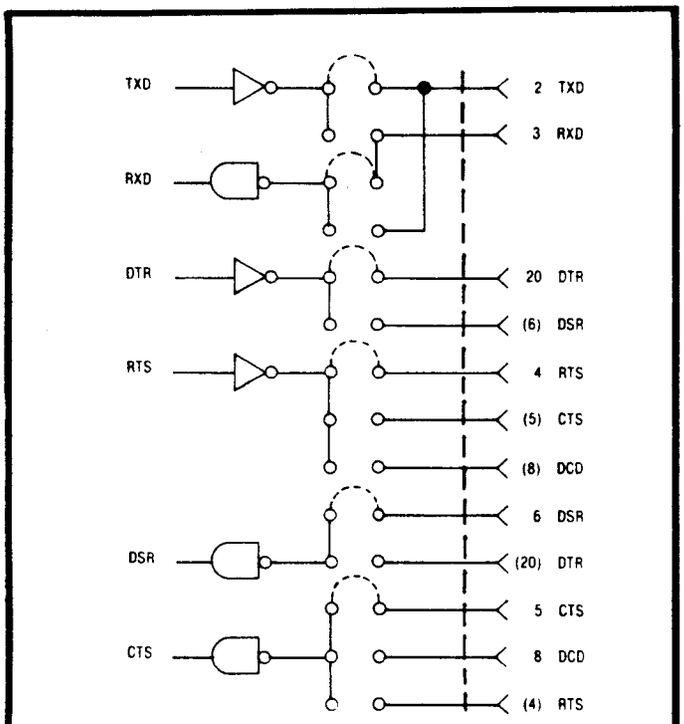


Figure 18: Full duplex general purpose jumperable as either DTE or (DCE.) for use with USARTs with only two control inputs.

Parts one and two of this series have examined the RS-232-C standard from the viewpoint of the microcomputer user. The standard was written long before the invention of the micro, so it takes a bit of shoe-horning to make it fit our needs. It is often mistakenly thought that this standard describes methods of encoding data to be transmitted. Not so; several other standards cover the ASCII code, start bits, stop bits, parity, synchronous and asynchronous techniques and handshaking protocols.

Part three of this series will describe methods for actually transmitting information over the TxD and RxD interchange circuits, and part four will present some of the integrated circuit chips needed to build a working serial interface.■

BUILD A HARDWARE PRINT SPOOLER

Part One: Background and Design

by Lance Rose, Technical Editor

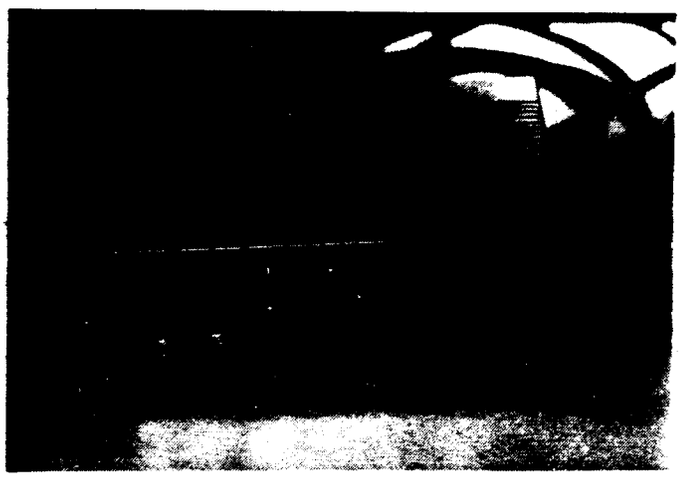
Most users of microcomputer systems would probably agree that printing hard copy is the slowest process occurring in their systems. Due to its highly mechanical nature, the printer simply can't keep up with the flow of data coming from electronic circuitry where processes occur in milli or microseconds. If you're like me, you've probably spent hours just watching your printer chug through a long program listing or print an endless series of statements or reports. With few exceptions, there simply isn't anything to do except watch the printer during these long outputs.

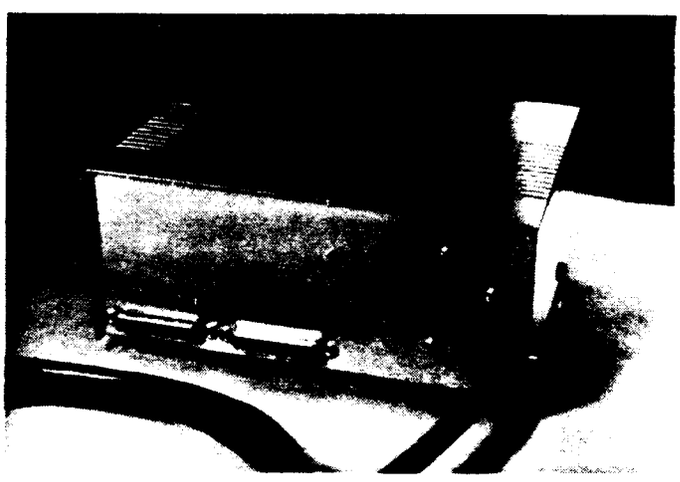
Since this isn't by any means a new or unique problem with computer systems, ways have been devised to keep the printer busy but still allow the user-operater to continue to do something useful with the computer while the printing process is going on. This is done by a method known as "spooling"

There are two general types of spooling used which I will call "software spooling"and "hardware spooling" . In the software version, instead of the computer sending each individual character to the printer, a slow process that occurs at printer speeds, the entire output to be printed is sent to a disk file known as the "spool file". Since writing to even a floppy disk is much faster than writing to a printer, this happens quite fast and the CPU is then free to perform some other task. But wait a minute. How does the output get from the disk file to the printer? This is done with some special software built into the operating system. This software knows when the spool file has something in it that needs to be sent to the printer. When this condition exists the software allows interrupts from the printer to occur whenever the printer is ready to accept a character. When each interrupt occurs, the interrupt handling routine retrieves one character from the spool file (actually from a buffer containing perhaps a sector at a time of the spool file) and sends it out to the printer. This takes very little time since the printer is already known to be ready and the CPU doesn't have to sit around waiting for this to happen. The result is that the time used for printing can overlap with useful time for doing another job with the system.

This method of spooling is widely used on mainframes and minicomputers where there is usually ample disk space (more often than not a hard disk or two) and where the operating systems include the necessary software to handle the spooling process. There is even a program available for CP/M® which will perform this process, albeit in a somewhat simplified manner, called DESPOOL® , available from Digital Research. Its use in microcomputers has been limited by the lack of true interrupt-driven operating systems as well as a lack of disk space in many systems thus denying the user a place to temporarily store large files to print.

The hardware spooling method is something that has become popular only in the last year or so in the microcomputer area. In this method the output is sent to a separate hardware print spooler which is most often simply a box containing a chunk of memory and a microprocessor. The communication to this device is performed at very high serial data transfer speeds (9600 or 19200 baud). The spooler performs two simultaneous functions. First, whenever a character is received from the computer it is input and stored temporarily in the spooler's RAM for later printing. Second, whenever the printer is ready and there is something in the RAM that should be printed, the spooler outputs this to the





printer. In addition, due to the fact that the memory of the spooler may be exceeded by the size of the printing job, the spooler must handshake with the computer and let it know when to stop sending characters. Similarly the spooler must be able to handshake in the other direction with the printer to keep from overflowing the printer's buffer in the case where the data transmission rate to the printer exceeds the actual physical printing speed. Since a number of handshaking conventions are in existence, the program used to run the spooler (contained in a ROM) must be able to determine or be preset to use a particular handshake convention. In the case of printers using the RS-232 serial interface standard, many use the DTR line (pin 20) to indicate a printer busy condition.

The main advantage of hardware spooling is that no changes to the software or operating system are necessary. As far as the computer is concerned, it is simply sending data to a very fast printer with a very large storage buffer. All that needs to be done is to reconfigure the serial port hardware for a faster baud rate than if it were communicating with the printer directly. Another advantage is that the method is not limited to any particular hardware or operating system. Any computer that has, for example, an RS-232 interface can output to a hardware spooler instead of a printer. The same would hold true if a Centronics interface were being used.

Of course there are some minor disadvantages to this spooling method. The only serious one is that there may be printing jobs that exceed the spooler memory size. If this happens then there is no appreciable speedup in printing since the computer must wait for the spooler to send some of the text to the printer before filling up its (the spooler's) memory again. This may occur a number of times before the last portion of the data to be printed is finally sent to the spooler. During the time that the spooler is emptying its memory to the printer, the computer is still waiting to send more output to the spooler and is thus prevented from beginning another task. Of course, since it is simple to provide a hardware spooler with up to 64K of RAM this should not be too much of a limiting factor except in the case of enormous printouts! In fact, in most applications much less than 64K of RAM can be used with a savings in expense. Most commercial spoolers on the market today start at 16K versions and go up to 64K in 16K steps. With a suitable design, a spooler can be built with as little as 1K or 2K RAM at a much lower cost.

So in fact, the main disadvantage of a hardware spooler, namely the limited RAM, can actually be an advantage if most printing jobs are relatively small allowing the construction of a smaller, less expensive piece of hardware. If it were absolutely necessary to design a hardware spooler with a capability of more than 64K, it would be possible to base it on one of the new 16-bit microprocessors that can address at least a megabyte of RAM. The cost of the microprocessor would not be too much more than that of say a Z80, but the additional cost of RAM would be substantial.

With all this in mind, I will present a design for a hardware print spooler that should be adequate to handle most printing jobs and allow simultaneous printing and computer use by the operator. Let me address each major point of the design separately:

(1) Microprocessor:

Although the program executed by whatever microprocessor is chosen will be relatively simple, in order to allow for upgrades the microprocessor should have a capable architecture. It should also be a low cost device and be in wide use. The Z80 fulfills these criteria and is widely available in different versions for as little as $5.

(2) Memory:

Here we have the choice of static vs. dynamic RAM. Each has its advantages. Dynamic memories are less expensive for the same storage capability and take up less board space for a full 64K. They are, however, more sensitive to noise on the power supply lines, require in most cases 3 supply voltages and are somewhat less reliable than their static counterparts. Static memories are easier to design with, more immune to noise and operate from a single supply. One other factor is that most inexpensive dynamic RAMs are available in a 16K or 64K x 1-bit architecture whereas static RAMs are available in 1-bit, 4-bit and 8-bit widths. The choice I have made here is the 6116 2K x 8-bit CMOS static RAM chip. Its architecture allows any size spooler to be built from 2K up to 62K (I'm allowing 2K for the program ROM). It has a low power consumption, is quite reliable and is easy to design with. Cost is somewhat more than dynamic RAM for a full 64K version but due to the fact that the dynamic RAMs need all the timing and control circuitry even for a small amount of actual memory, a spooler with a small or moderate amount of memory should cost the same or less to build with static RAM than with dynamic. I have estimated the crossover point at about 32K both in cost and in board layout space so that is the size I have chosen to present in this series of articles.

Although the EPROM type is not too important, the 2716 has virtually the same pinout as the 6116 RAM chip so the chip select logic is simplified if it is used.

(3) Interface:

There are a number of interface standards in use today: RS-232, Centronics, IEEE-488 to name just a few. I chose the RS-232 interface to use in this design simply because most of the letter quality printers I work with use it and I'm more familiar with it than any other. It may be the most widely used standard but I'm not aware of any statistics to that effect. I'm assuming a DTR handshaking protocol here, that is, pin 20 is used to signal a printer busy condition by going to a logic low state (approximately -12 volts). This will be used both by the printer to tell the spooler to stop sending, and by the spooler to tell the computer likewise. In the last part of this series of articles I will show how to modify the spooler to use a Centronics interface or software handshaking (ETX/ACK or X-on/X-off). That will also allow interface conversion to occur during the spooling process. For example a computer with only a parallel Centronics interface could still send output to an RS-232 printer via a spooler with a

continued on p. 15

A REVIEW OF FLOPPY DISK FORMATS

by M. Mosher

Whenever the subject of software exchange comes up, as it often does, the question arises of "Why can't I just take my diskette from system A and put it into a drive on system B and have it work?" To answer this question a discussion of the differences in floppy disk format "standards" is in order. What I'll do here is take the characteristics of a floppy disk one at a time and point out the similarities and differences.

Size

This one is pretty obvious. If you try to put a 5.25-inch diskette into an 8-inch slot it's going to just flop around in there (no pun intended). Conversely you just aren't going to fit an 8-inch diskette into a 5.25-inch slot at all unless you use a pair of scissors and I'm not even going to begin to address that issue. To add to the variation, Sony has recently introduced a 3-inch "microfloppy" drive which should be entering production very soon.

Number of Tracks

Most manufacturers have pretty much standardized on this parameter though there are some variations. Full-size (8-inch) floppies almost always have 77 tracks to the diskette, minifloppies (5.25-inch) have mostly had 35 tracks in the past but many are showing up now with 40 tracks. Of course a floppy that has 40 tracks of data on it can't be read on a system whose hardware can only read 35 tracks from the diskette. Yet another variation are floppy drives whose tracks are packed twice as closely on the diskette allowing 80 tracks on a minifloppy.

Number of Sides

Early floppies used only one side of the diskette to record data on, leaving the second side blank. Many hobbyists saw this as a waste and took to punching another hole and write-protect notch in the diskette jacket to be able to use the second side of the diskette as well (most diskettes have a magnetic coating on the second side as well as the first). To get at the second side, however, you have to remove the diskette from the drive, turn it over and re-insert it into the slot. More recent drives have a second head to read the second side, making it unnecessary to modify the diskette itself or turn it over to get at the data on the second side. Something to watch out for here — a double-sided diskette made on a true double-sided drive won't work on a single-sided drive by just turning the diskette over and trying to read the second side. The sense of rotation is opposite in each case. Think about it for a while.

Sectoring

This takes a little explanation. Within each track the data is subdivided into "sectors", a sector being simply a fraction of the total track. One obvious variable is simply the number of sectors a track is divided into. Various disk formats have anywhere from 8 to 32 sectors per track. Since the sectors may be different sizes (anywhere from 128 to 1024 bytes per sector) this introduces incompatibilities.

A second aspect of this is in the form the sectoring may take — either "hard" or "soft." In hard sectoring the beginning of each sector on the diskette is marked by a small hole punched in the diskette near the large center hole. As the disk rotates, these small holes pass under a light source with a photodetector on the opposite side of the diskette. A short electrical pulse is generated by the photodetector as each hole exposes the light source. This indicates to the computer that the beginning of a sector is present. One additional hole called the index hole is punched midway between two of the sectors. A pulse coming halfway between two sectors tells the computer that the next sector pulse will be the first one on the track.

In soft sectoring quite a different method is used. Only a single hole is punched in the diskette — the index hole. It tells the disk controller that the track begins immediately. Here, however the beginning of each sector and the boundaries between them are actually written onto the track as information. The computer finds a sector by reading the track continuously until it comes to a "header"(a short piece of coded information) that indicates the start of the desired sector. It can then begin to read the actual data contained in that sector.

Both types of sectoring have their advantages and disadvantages. In hard sectoring the diskettes can usually be taken and used immediately without the need for preparing or "formatting" them. Hard sectoring also is usually a little simpler than soft sectoring since all the circuitry has to do is detect a pulse rather than decode header information. This is really a minor difference though. A more important advantage is that without the need for sector headers, more space on the track can be allocated to storing actual data.

Although soft sectoring has the additional overhead of sector headers with the need to format a diskette before using it (formatting simply writes the sector headers onto each track), it does have some advantages. By detecting sectors by reading the header which contains, among other things, the track number and sector number, the computer can verify that it is on the correct track and reading the correct sector. This usually isn't done with hard sectoring and

provides an additional protection against errors. Also, since the sector boundaries (and thus sizes) are recorded in the same way as data on the track, the boundaries can be almost anywhere, thus allowing a variety of sector sizes and number of sectors per track. This flexibility can sometimes be useful.

Density

This has to do with the amount of data that can be packed onto a given space on the diskette. Originally all floppies used single-density encoding methods (also known as FM encoding). In this the data pulses and clock pulses are combined and both recorded onto the diskette surface. When read back, the clock and data are separated by appropriate circuity and the latter passed on to the CPU. To keep up nwith the demand for larger databases and such, other encoding methods have been developed to pack more information into the same space on the diskette. In double-density encoding (known also by the term MFM), the data is written onto the diskette without any clock pulses. This in effect allows each pulse on the track to be a data bit rather than alternating data with clock. The only problem here is that when the data is read back in, the clock pulses must be resurrected from the data. How this is done is beyond the scope of this article but suffice it to say it can be done, but with some difficulty relative to single-density encoding. This makes the timing requirements and disk rotational stability more critical in double density, but with development it has become quite reliable and many disk systems sold today are capable of recording in both single and double density.

In Summary

I think you can see by now that there are a lot of variables involved in diskette formats. If we take the three possible sizes, three different values of tracks-per-diskette, two possibilities for number of sides, two values for type of sectoring, perhaps five different values for number of sectors per track and two different densities, we have something like $3 \times 3 \times 2 \times 2 \times 5 \times 2 = 360$ different formats that are possible. Although in practice the situation isn't this bad there are at present maybe a dozen different diskette formats in popular use. So the next time you wonder why your Apple diskette won't work in a Radio Shack or S-100 machine, just realize that it only has one chance in 360 of doing so; something like ⅓ of one percent. ■

"Build a Print Spooler," continued from p. 13

Centronics input and an RS-232 output. Other conversions would be possible, too.

4) Serial Communications IC:

Having chosen the Z80 for a microprocessor, there are several choices for a serial I/O chip. One is the Z80 SIO. However, it is an expensive chip and is so flexible as to be confusing to the average user. The Intel 8251 is cheaper and not as complicated but still requires some understanding. The variety of UARTs available are the least expensive, require no software initialization and are adequate for the task here. They are readily available from a number of sources for $4 and up. Needless to say, the UART was chosen here.

(5) Support Circuitry:

The choices here may not be so clear-cut. Since I live in an area where it's not possible to walk or drive down to the corner chip shop for something I may need, I tend to design most circuits around common, easily available chips. Most chips used in this circuit are available, if absolutely necessary, at Radio Shack. In addition I believe simplicity to be a virtue and try to design accordingly.

I think you can see some of the reasons for my design here. Not all would necessarily agree with everything I've said but I can say that this design approach leads to a circuit that is fairly easy to build and troubleshoot and works well when complete.

In Part 2 I will present the hardware construction layout and schematics for the spooler along with some suggestions for a power supply and case to put it in. Also I'll give a flowchart and listing for the spooler's operating program along with some additional comments on the software. ■

Correction

The September **Computer Hacker** contained an error in the RS-232-C article. On page 4, in the section titled "Hacker's View of the Mechanical Requirements," the first requirement reads "The DTE must provide a female connector..." The sentence should read "The DTE must provide a **male** connector..."

We regret any inconvenience this error may have caused. Please don't hesitate to write if you find something which you believe to be an error.

SENDING MORSE CODE WITH AN APPLE][

by Marvin L. De Jong

Introduction

Using a computer to send Morse code is a clean, well-defined programming problem, and it has always been one of my favorite real-time control applications. Receiving Morse code with the aid of a computer is a more difficult task, especially if any serious attempt is made to approach the capability of a human being using a modern communications receiver. The latter problem is not associated with the computer or the program, but rather with the analog circuitry that converts the tones into logic levels. In this article we will confine ourselves to the problem of sending Morse code, a task for which a machine can easily outperform a human being.

Program Features

1. Morse code can be sent from the Apple keyboard at rates, selected from the keyboard, from 8 to 100 wpm (words per minute.)

2. A ring buffer allows the typist to type up to 225 characters ahead of the one being sent.

3. Three messages, totalling 256 characters, may be stored and sent with commands from the keyboard. Characters from the keyboard may be inserted in these messages as they are being sent, a desirable feature for contest operation.

4. The computer can also be operated as an electronic keyer that operates at the speed entered on the keyboard.

5. In its keyer mode the program reads what is sent and prints it on the video monitor. You can use this feature to monitor what you are sending, provided you send it correctly.

6. The Apple][speaker provides a sidetone, making the program useful for code practice.

7. The game i/o connector is used to interface the computer to the transmitter with simple components.

Hardware

The hardware required to use the program consists of a simple interface between the game i/o connector and the transmitter, a 1500 Hz source of interrupts, and a simple keyer interface if you wish to use the program in the keyer mode. The circuit to key the transmitter is shown in figure 1. The optional keyer circuitry is shown in Figure 2. As far as a source of interrupts is concerned, we used a John Bell Engineering 6522 board in slot seven.

The interrupts occur at a 1500 Hz rate. For those who are inclined to build circuits, a less expensive source of interrupts is a 555 timer, multivibrating at 1500 Hz, and connected to a 74LS121 one-shot wired to produce a 10 microsecond logic-zero pulse on the IRQ line. The IRQ line can be accessed on a peripheral card connector. The program initializes the John Bell 6522 card for proper operation, but the program is transparent to the source of interrupts. It is important that they occur at a 1500 Hz rate. The program assumes the John Bell card is in slot seven, the 6522 labelled U1 is used, and a jumper is added to the card to connect the IRQ on the 6522 to the IRQ line in the Apple. Holes on the card are provided for this jumper.

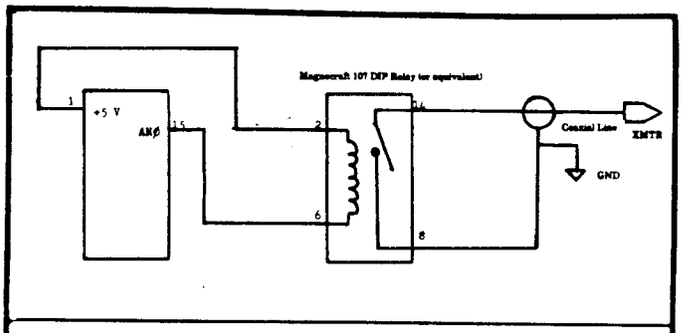


Figure 1
A circuit to key a transmitter from the game i/o connector.

Program Operation

Load all the programs in the listings. Type in RUN and press the RETURN key. The computer responds by requesting the code speed. Enter this and then press return. The screen will go blank and you can start typing. Type some letters, numbers, and punctuation marks. You should hear Morse code coming from the speaker. If there is a problem, check your disassembled version of the program against the listings. Also make sure that the MORSE TABLE and the ASCII TABLE are loaded. Asssuming that everything is running correctly, you can practice sending at the keyboard. The reverse arrow key allows you to delete characters entered in the buffer provided they have not yet been sent. Try typing ahead, then delete some characters with the reverse arrow key.

To change code speeds simply press the ESC key and the program will return to the BASIC routine to allow you to enter a new speed.

To load messages press CTRL L. Type in message A. For example, message A might be CQ CQ CQ DE KOEI KOEI K. When message A is complete type RETURN. Now enter message B followed by RETURN, and then enter message C followed by RETURN. Now you are back in the code sending mode.

To send message A,B, OR C, simply type CTRL A, CTRL B, or CTRL C. ANy message may be interrupted from the keyboard, but you must be alert. It will help to insert an extra space or two in the message where you with to interrupt it.

To use the program as a keyer you must construct the

circuit in Figure 2 and make the connections to the game i/o connector. Try this and see how you like the keyer operation. Note that what you send is what you see. The program converts your characters from Morse to alphanumeric characters on the video monitor.

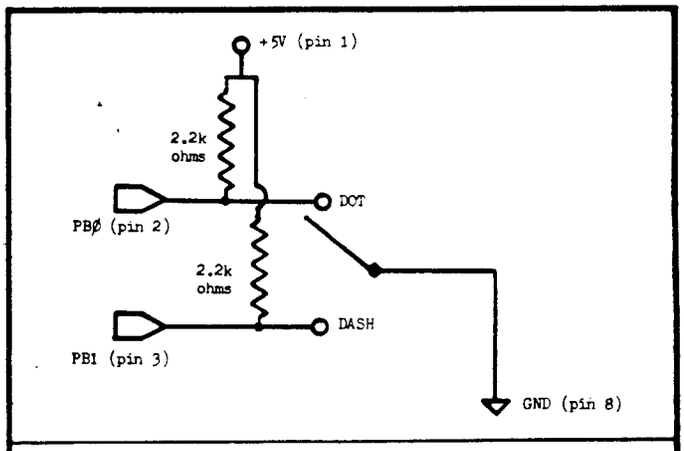


Figure 2
Circuit diagram of the keyer. Pin numbers refer to the Apple][game i/o connector. My circuit worked very well without the more or less standard pull-up resistors (2.2 kohm.)

The listings have extensive comments, enough to make the program understandable. During each interrupt the keyboard is tested to see if a character has been entered. If a dot or dash is being sent the speaker is togged to produce a 750 Hz tone. Two counters are incremented or decremented to keep track of the number of 1500 Hz pulses that have occured. The pushbutton inputs, PB0 and PB1, are tested to see if the keyer is being used. Various bits in a register called FLAG are set or cleared depending on which events occur: key down, PB1 at logic zero, speaker to be toggled, etc. This memory location is then analyzed by the main program so that it can take the appropriate action. Once the flow of the action is appreciated, the comments, labels and subroutine names should make the operation understandable.

Listing 1. The BASIC Driver Routine.

```
]LIST

5  REM  MORSE DRIVER ROUTINE
10  POKE 10,76: POKE 11,00: POKE 12,17
20  PRINT "AT WHAT SPEED WOULD YOU LIKE TO SEND?"
30  PRINT "TYPE A NUMBER BETWEEN 0 AND  100, THEN PRESS RETURN."
40  INPUT SPEED
50 DOT = 1B00 / SPEED
60  POKE 6,DOT
70 Y =  USR (0)
80  GOTO 20
90  END
```

Listing 2. The MAIN PROGRAM.

```
:ASM

                1    TILL     EQU  $C704
                2    TILH     EQU  $C705
                3    ACR      EQU  $C70B
                4    IER      EQU  $C70E
                5    SNDKEY   EQU  $1009
                6    FIFO     EQU  $EB
                7    PNTR     EQU  $ED
                8    FLAG     EQU  $07
                9    SPCFLG   EQU  $08
               10    TIMOUT   EQU  $06
               11    COUT     EQU  $FDF6
               12    ASCII    EQU  $0EB0
               13    CHAR     EQU  $EF
               14    DOT      EQU  $102B
               15    DASH     EQU  $104E
               16    CNTR2    EQU  $09
               17    CNTR1    EQU  $CE
               18    HOME     EQU  $FC58
               19    SNDABC   EQU  $1094
               20    LOADMS   EQU  $105A
               21             ORG  $1100
1100: 78       22    MAIN     SEI               ;PREVENT INTERRUPTS
1101: 8D 59 C0 23             STA  $C059        ;TURN RELAY OFF
1104: D8       24             CLD
1105: A9 FF    25             LDA  #$FF
1107: 8D 04 C7 26             STA  TILL
110A: A9 02    27             LDA  #$02
110C: 8D 05 C7 28             STA  TILH
110F: A9 40    29             LDA  #$40
1111: 8D 0B C7 30             STA  ACR
1114: A9 C0    31             LDA  #$C0
1116: 8D 0E C7 32             STA  IER          ;ENABLE INTERRUPTS FROM T1
1119: 20 58 FC 33             JSR  HOME         ;CLEAR SCREEN
111C: A9 00    34             LDA  #$00         ;SET UP THE INTERRUPT VECTOR
111E: 8D FE 03 35             STA  $03FE
1121: A9 12    36             LDA  #$12
1123: 8D FF 03 37             STA  $03FF
1126: A9 00    38             LDA  #00          ;CLEAR VARIOUS REGISTERS
1128: 85 EB    39             STA  FIFO
112A: 85 ED    40             STA  PNTR
112C: 85 09    41             STA  CNTR2
112E: 85 CE    42             STA  CNTR1
1130: 85 07    43             STA  FLAG
1132: 85 08    44             STA  SPCFLG
1134: A9 01    45             LDA  #01
1136: 85 EF    46             STA  CHAR
1138: A9 0A    47             LDA  #$0A
113A: 85 EE    48             STA  PNTR+1
113C: 85 EC    49             STA  FIFO+1
113E: A0 00    50             LDY  #00
1140: 58       51             CLI               ;ALLOW INTERRUPTS
1141: A5 EB    52    BRB      LDA  FIFO

1143: C5 ED    53             CMP  PNTR
1145: F0 0B    54             BEQ  DOWN
1147: 20 09 10 55             JSR  SNDKEY
114A: B8       56             CLV
114B: 50 F4    57             BVC  BRB
114D: 24 07    58    DOWN     BIT  FLAG         ;CONTROL CHARACTER?
114F: 10 2A    59             BPL  DOWN1        ;NO.
1151: A9 7F    60             LDA  #$7F         ;CLEAR CONTROL FLAG
1153: 25 07    61             AND  FLAG
1155: 85 07    62             STA  FLAG
1157: AD 00 C0 63             LDA  $C000        ;READ KEYBOARD AGAIN
115A: 8D 10 C0 64             STA  $C010        ;CLEAR STROBE
115D: C9 9B    65             CMP  #$9B         ;ESCAPE?
115F: D0 02    66             BNE  BR6
1161: 78       67             SEI               ;PREVENTS INTERRUPTS WHILE
                                                  IN BASIC
1162: 60       68             RTS
1163: C9 84    69    BR6      CMP  #$84         ;CONTROL A,B,OR C?
1165: B0 0B    70             BCS  BR5
1167: 29 7F    71             AND  #$7F
1169: 20 94 10 72             JSR  SNDABC       ;SEND THE MESSAGE
116C: B8       73             CLV
116D: 50 D2    74             BVC  BRB
116F: C9 8C    75    BR5      CMP  #$8C
1171: D0 CE    76             BNE  BR8
1173: 78       77             SEI               ;PREVENT INTERRUPTS WHILE
1174: 20 5A 10 78             JSR  LOADMS       ;LOADING MESSAGES
1177: 58       79             CLI               ;ALLOWS INTERRUPTS AGAIN
1178: B8       80             CLV
1179: 50 C6    81    BACK     BVC  BRB
117B: A9 01    82    DOWN1    LDA  #$01         ;CHECK DOT FLAG
117D: 24 07    83             BIT  FLAG
117F: F0 11    84             BEQ  PAST         ;NO DOT
1181: 20 2B 10 85             JSR  DOT          ;SEND A DOT AND A SPACE
1184: 06 EF    86             ASL  CHAR         ;SHIFT THE CHARACTER REGISTER
1186: C6 07    87             DEC  FLAG         ;CLEAR THE DOT FLAG
1188: A9 00    88    UP       LDA  #00          ;CLEAR THE COUNTER
118A: 85 CE    89             STA  CNTR1
118C: A9 01    90             LDA  #01          ;SET SPACE FLAG
118E: 85 08    91             STA  SPCFLG
1190: D0 AF    92    STEP     BNE  BRB          ;FORCE A JUMP BACK TO BR8
1192: A9 02    93    PAST     LDA  #02          ;CHECK THE DASH FLAG
1194: 24 07    94             BIT  FLAG
1196: F0 0F    95             BEQ  OUTPUT
1198: 20 4E 10 96             JSR  DASH         ;SEND A DASH
119B: A9 FD    97             LDA  #$FD         ;CLEAR  DASH FLAG
119D: 25 07    98             AND  FLAG
119F: 85 07    99             STA  FLAG
11A1: 06 EF    100            ASL  CHAR
11A3: E6 EF    101            INC  CHAR
11A5: D0 E1    102            BNE  UP           ;JUMP UP, THEN BACK TO BR8
11A7: A5 08    103   OUTPUT   LDA  SPCFLG       ;CHECK ON SPACES
11A9: F0 96    104            BEQ  BR8          ;NOT TIME FOR A CHARACTER
11AB: A5 CE    105            LDA  CNTR1

11AD: C5 06    106            CMP  TIMOUT       ;HAS ONE DOT TIME PASSED?
11AF: 90 90    107            BCC  BR8          ;NO
11B1: A5 08    108            LDA  SPCFLG       ;YES, OUTPUT CHARACTER OR SPACE?
11B3: C9 01    109            CMP  #01
11B5: D0 16    110            BNE  WDSP
11B7: E6 08    111            INC  SPCFLG
11B9: A6 EF    112            LDX  CHAR         ;GET CHARACTER
11BB: BD B0 0E 113            LDA  ASCII,X      ;LOOK UP ASCII REPRESENTATION
11BE: 09 80    114            ORA  #$80         ;SET BIT SEVEN
11C0: 20 F6 FD 115            JSR  COUT         ;OUTPUT IT
11C3: A9 00    116            LDA  #00
11C5: 85 CE    117            STA  CNTR1
11C7: A9 01    118   HERE     LDA  #01
11C9: 85 EF    119            STA  CHAR         ;RESET CHARACTER REGISTER
11CB: D0 C3    120            BNE  STEP
11CD: A5 08    121   WDSP     LDA  SPCFLG       ;CHECK SPACE FLAG AGAIN
11CF: C9 04    122            CMP  #04
11D1: F0 0B    123            BEQ  OUTPUT2
11D3: A9 00    124            LDA  #00          ;CLEAR CNTR1
11D5: 85 CE    125            STA  CNTR1
11D7: E6 08    126            INC  SPCFLG
11D9: D0 B5    127            BNE  STEP
11DB: A9 A0    128   OUTPUT2  LDA  #$A0         ;OUTPUT A SPACE
11DD: 20 F6 FD 129            JSR  COUT
11E0: A9 00    130            LDA  #00          ;CLEAR SPACE FLAG
11E2: 85 08    131            STA  SPCFLG
11E4: F0 E1    132            BEQ  HERE         ;CLOSE THE LOOP
               133   Q

--End assembly--

238 bytes

Errors: 0
```

continued

Listing 3. The SUBROUTINES.

```
:ASM

                        1   TIMOUT  EQU  $06
                        2   FLAG    EQU  $07
                        3   CNTR2   EQU  $09
                        4   CNTR1   EQU  $CE
                        5   FIFO    EQU  $EB
                        6   PNTR    EQU  $ED
                        7   ABCBUF  EQU  $0200
                        8   TEMP    EQU  $FF
                        9   START   EQU  $F9
                        10  END     EQU  $FC
                        11  COUT    EQU  $FDF6
                        12  RDKEY   EQU  $FD0C
                        13  CODE    EQU  $0C00
                        14          ORG  $1000
                        15
1000: A5 06             16  TIMER   LDA  TIMOUT     ;START COUNTING DOWN
1002: 85 09             17          STA  CNTR2      ;IN INTERRUPT ROUTINE
1004: A5 09             18  WAIT    LDA  CNTR2
1006: D0 FC             19          BNE  WAIT
1008: 60                20          RTS
                        21
1009: A0 00             22  SNDKEY  LDY  #$00       ;Y=0 TO READ RING BUFFER
100B: B1 ED             23          LDA  (PNTR),Y   ;GET A CHARACTER
100D: E6 ED             24          INC  PNTR       ;UPDATE POINTER TO RING BUFFER
100F: AA                25  ENTRY   TAX             ;ASCII TO X REGISTER
1010: BD 00 0C          26          LDA  CODE,X     ;TO LOOK UP MORSE CODE
1013: F0 41             27          BEQ  WDSPCE     ;ZERO IS A WORD SPACE
1015: 8D 00 0C          28          STA  CODE       ;STORE CHARACTER
1018: 0E 00 0C          29  REST    ASL  CODE       ;SHIFT IT INTO CARRY
101B: F0 35             30          BEQ  CHSPCE     ;ZERO MEANS CHARACTER IS SENT
101D: B0 06             31          BCS  DAH        ;CARRY SET IMPLIES DASH
101F: 20 2B 10          32          JSR  DOT        ;OTHERWISE SEND A DOT
1022: B8                33          CLV             ;FORCE A BRANCH BACK TO GET
1023: 50 F3             34          BVC  REST       ;THE REST OF THE CHARACTER
1025: 20 4E 10          35  DAH     JSR  DASH       ;SEND A DASH
1028: B8                36          CLV             ;FORCE A BRANCH BACK
1029: 50 ED             37          BVC  REST
                        38
102B: A2 01             39  DOT     LDX  #01        ;X IS NUMBER OF DOTS
102D: 8E 58 0C          40  HERE    STX  $0C058     ;TURN RELAY ON
1030: A9 40             41          LDA  #$40       ;SET UP MASK FOR FLAG
1032: 05 07             42          ORA  FLAG
1034: 85 07             43          STA  FLAG       ;SET SPEAKER BIT IN FLAG
1036: 20 00 10          44  BACK    JSR  TIMER      ;WAIT FOR ONE DOT TIME
1039: CA                45          DEX
103A: D0 FA             46          BNE  BACK
103C: 8E 59 0C          47          STX  $0C059     ;TURN RELAY OFF
103F: A9 BF             48          LDA  #$BF       ;CLEAR SPEAKER BIT IN FLAG
1041: 25 07             49          AND  FLAG
1043: 85 07             50          STA  FLAG
1045: A2 01             51  SPACE   LDX  #$01      ;ADD A SPACE
1047: 20 00 10          52  MORE    JSR  TIMER
104A: CA                53          DEX
104B: D0 FA             54          BNE  MORE
104D: 60                55          RTS             ;SPACE COMPLETE
104E: A2 03             56  DASH    LDX  #$07       ;DASH IS THREE DOTS
1050: D0 DB             57          BNE  HERE
1052: A2 02             58  CHSPCE  LDX  #02        ;CHARACTER SPACE
1054: D0 F1             59          BNE  MORE
1056: A2 04             60  WDSPCE  LDX  #04        ;WORD SPACE
1058: D0 ED             61          BNE  MORE
                        62
105A: A2 01             63  LOADMS  LDX  #01
105C: A0 00             64          LDY  #00
105E: 94 F8             65  NEXT    STY  START-1,X  ;STARTING INDEX FOR MESSAGES
1060: 98                66  OVER    TYA
1061: 48                67          PHA             ;SAVE Y ON THE STACK
1062: 20 0C FD          68          JSR  RDKEY      ;GET A CODE FROM THE KEYBOARD
1065: 20 F6 FD          69          JSR  COUT       ;OUTPUT IT TO THE MONITOR
1068: C9 BB             70          CMP  #$BB       ;WAS IT A BACKSPACE?
106A: D0 0A             71          BNE  BR1        ;NO.
106C: 68                72          PLA             ;YES, GET Y BACK
106D: A8                73          TAY
106E: B8                74          DEY             ;DELETE THE CHARACTER BY LOADING
106F: A9 C0             75          LDA  #$C0       ;THE BUFFER WITH A SPACE
1071: 99 00 09          76          STA  ABCBUF,Y
1074: D0 EA             77          BNE  OVER       ;FORCE A JUMP TO GET A NEW KEY
1076: C9 BD             78  BR1     CMP  #$BD       ;WAS IT A "RETURN"?
1078: F0 0C             79          BEQ  PAST       ;YES. END THE MESSAGE
107A: 85 FF             80          STA  TEMP       ;STORE CHARACTER FOR A MOMENT
107C: 68                81          PLA             ;GET Y BACK
107D: A8                82          TAY
107E: A5 FF             83          LDA  TEMP       ;GET CHARACTER BACK
1080: 99 00 09          84          STA  ABCBUF,Y   ;STORE IT IN THE MESSAGE BUFFER
1083: C8                85          INY
1084: D0 DA             86          BNE  OVER       ;GO BACK FOR ANOTHER CHARACTER
1086: 68                87  PAST    PLA
1087: A8                88          TAY
1088: B8                89          DEY             ;STORE INDEX FOR THE END OF
1089: 94 F8             90          STY  END-1,X    ;EACH MESSAGE
108B: C8                91          INY
108C: F0 05             92          BEQ  OUT        ;GET OUT IF Y=0
108E: E8                93          INX             ;GET ANOTHER MESSAGE UNLESS
108F: E0 04             94          CPX  #04        ;WE HAVE THREE ALREADY
1091: 90 CB             95          BCC  NEXT       ;INPUT THE NEXT MESSAGE
1093: 60                96  OUT     RTS             ;ALL THE MESSAGES ARE IN MEMORY
                        97
1094: AA                98  SNDABC  TAX             ;CONTROL KEY CODE TO X
1095: B4 F8             99          LDY  START-1,X  ;PICK UP THE STARTING INDEX
1097: 8A                100 LOOP    TXA             ;SAVE X
1098: 48                101         PHA             ;ON THE STACK
1099: B9 00 09          102         LDA  ABCBUF,Y   ;FETCH THE MESSAGE
109C: 20 0F 10          103         JSR  ENTRY      ;SEND A CHARACTER
109F: 98                104         TYA             ;SAVE Y
10A0: 48                105         PHA
10A1: A5 EB             106 BR9     LDA  FIFO       ;NEED TO SEND A CHARACTER FROM
10A3: C5 ED             107         CMP  PNTR       ;THE RING BUFFER?
10A5: F0 06             108         BEQ  BR10       ;NO
10A7: 20 09 10          109         JSR  SNDKEY     ;YES
10AA: B8                110         CLV
10AB: 50 F4             111         BVC  BR9
10AD: 68                112 BR10    PLA             ;GET Y BACK
10AE: A8                113         TAY
10AF: 68                114         PLA             ;GET X
10B0: AA                115         TAX             ;FROM THE STACK
10B1: 98                116         TYA             ;END OF MESSAGE?
10B2: D5 FB             117         CMP  END-1,X
10B4: B0 03             118         BCS  FINISH     ;YES.SO QUIT
10B6: C8                119         INY             ;NO,GET ANOTHER CHARACTER
10B7: 90 DE             120         BCC  LOOP
10B9: 60                121 FINISH  RTS

--End assembly--

186 bytes

Errors: 0
```

Listing 4. The INTERRUPT Routine.

```
:ASM

                        1   FLAG    EQU  $07
                        2   TICL    EQU  $C704
                        3   PB0     EQU  $C061
                        4   PB1     EQU  $C062
                        5   FIFO    EQU  $EB
                        6   PNTR    EQU  $ED
                        7   COUT2   EQU  $FDF6
                        8   CNTR1   EQU  $CE
                        9   CNTR2   EQU  $09
                        10          ORG  $1200
1200: 98                11  IRQRTN  TYA
1201: 48                12          PHA
1202: AD 04 C7          13          LDA  TICL       ;CLEAR T1 INTERRUPT FLAG
1205: 24 07             14          BIT  FLAG       ;TEST THE FLAG
1207: 50 03             15          BVC  BR1        ;SPEAKER FLAG OFF
1209: AD 30 C0          16          LDA  $C030      ;TOGGLE SPEAKER
120C: E6 CE             17  BR1     INC  CNTR1      ;INCREMENT COUNTER ONE
120E: C6 09             18          DEC  CNTR2      ; DECREMENT COUNTER TWO
1210: AD 00 C0          19          LDA  $C000      ;READ KEYBOARD
1213: 10 3F             20          BPL  BR2        ;NO KEY
1215: C9 A0             21          CMP  #$A0       ;IS IT A CONTROL CHARACTER?
1217: 90 1B             22          BCC  BR3        ;YES
1219: A0 00             23          LDY  #00
121B: 91 EB             24          STA  (FIFO),Y   ;STORE THE CHARACTER IN THE
121D: E6 EB             25          INC  FIFO       ;RING BUFFER
121F: 20 F6 FD          26  BACK    JSR  COUT2      ;OUTPUT THE CHARACTER
1222: 8D 10 C0          27  HERE    STA  $C010      ;CLEAR STROBE
1225: A4 24             28          LDY  $24        ;ADVANCE THE CURSOR
1227: B1 28             29          LDA  ($28),Y
1229: 29 3F             30          AND  #$3F
122B: 09 40             31          ORA  #$40
122D: 91 28             32          STA  ($28),Y
122F: 68                33  OUT     PLA             ;GET Y FROM THE STACK
1230: A8                34          TAY
1231: A5 45             35          LDA  $45        ;GET A FROM MEMORY
1233: 40                36          RTI             ;RETURN
1234: C9 BB             37  BR3     CMP  #$BB       ;DELETE KEY?
1236: D0 0E             38          BNE  NEXT1
1238: 20 F6 FD          39          JSR  COUT2
123B: A5 EB             40          LDA  FIFO
123D: C5 ED             41          CMP  PNTR
123F: F0 E1             42          BEQ  HERE
1241: C6 EB             43          DEC  FIFO
1243: B8                44          CLV
1244: 50 DC             45          BVC  HERE
1246: C9 8D             46  NEXT1   CMP  #$8D       ;CARRIAGE RETURN
1248: D0 02             47          BNE  NEXT2
124A: F0 D3             48          BEQ  BACK
124C: A9 80             49  NEXT2   LDA  #$80
124E: 05 07             50          ORA  FLAG
1250: 85 07             51          STA  FLAG
1252: D0 DB             52          BNE  OUT
1254: A5 07             53  BR2     LDA  FLAG       ;CHECK THE DOT AND DASH FLAGS
1256: 29 03             54          AND  #03        ;BET OUT IF THEY ARE SET
1258: D0 D5             55          BNE  OUT
125A: 2C 61 C0          56          BIT  PB0        ;READ THE PUSH BUTTON
125D: 30 04             57          BMI  NEXT3      ;INPUTS
125F: E6 07             58          INC  FLAG       ;SET THE DOT FLAG
1261: D0 CC             59          BNE  OUT
1263: 2C 62 C0          60  NEXT3   BIT  PB1        ;CHECK FOR A DASH
1266: 30 C7             61          BMI  OUT
1268: A9 02             62          LDA  #02
126A: 05 07             63          ORA  FLAG       ;SET THE DASH FLAG
126C: 85 07             64          STA  FLAG
126E: D0 BF             65          BNE  OUT

--End assembly--

112 bytes

Errors: 0
```

Morse Table

```
0C80- 00 00 00 00 00 00 00 CE
0C88- 00 00 00 00 CE 8C 56 94
0C90- FC 7C 3C 1C 0C 04 84 C4
0C98- E4 F4 16 32 00 8C 00 32
0CA0- 00 60 88 A8 90 40 28 D0
0CA8- 08 20 16 32 CE 8C 56 94
0CB0- FC 7C 3C 1C 0C 04 84 C4
0CB8- E4 F4 16 32 20 8C 20 32
0CC0- 00 60 88 A8 90 40 28 D0
0CC8- 08 20 78 B0 48 E0 A0 F0
0CD0- 68 D8 50 10 C0 30 18 70
0CD8- 98 B8 C8 00 00 00 00 00
0CE0- 00 00 00 00 00 00 00 00
0CE8- 00 00 00 00 00 00 00 00
0CF0- 00 00 00 00 00 00 00 00
0CF8- 00 00 00 00 00 00 00 00
```

continued on p.22

Beginner's Column, Part Two:

ANYONE FOR A LITTLE "KISS" ELECTRONICS?

by Phil Wells, Technical Editor

How much electronics theory do you have to know to be able to design your own computer-related projects? A dozen or so basic concepts and formulas will get you started. Beyond that, one of the great things about this hobby is that you can dig into theory just as far (or as little) as you want. It just helps a lot to be able to learn it "hands-on." That's what this column is for.

As discussed in last month's KISS, you will need at least a VOM (Volt-Ohm-Milliammeter) and some small tools and parts. I'm using a Radio Shack #22-204 multitester and a Radio Shack #22-191 digital multimeter. These are not the best but are widely available, very low cost, and have worked well for me for several years.

Electronics at our level is all about what happens when we push electrons through circuits.

We will talk about simplifications of the real world, make calculations based on idealized components, then construct real circuits and make measurements to test our simplified models. What we care about is being able to put together a project that does something useful or interesting. You should understand from the beginning that real components won't always match our simplified models, that real measuring devices have built-in sources of errors and that most of the time a measured value that comes close to our calculated value is a success. Don't expect a 4700 Ohm resistor to measure exactly 4700 Ohms, and don't waste time trying to get 5.000 volts when we need 4.8 to 5.2 volts.

Getting Started: Ohm's Law

We can easily measure current, voltage and resistance. These are most beautifully related by Ohm's Law (figure 1). This formula says that if we connect a one ohm resistor across an ideal one volt battery, one ampere of current will flow through the resistor (figure 2).

The battery supplies electrons, each carrying one negative electrical charge. A battery is a chemical device which produces a potential difference, or voltage. The potential difference represents an ability to do work. The work is performed by moving charges from one side of the battery to the other, through a conductor connecting the two terminals. If there is no conductor, no work is performed, but the potential remains. When the battery runs out of charged particles, there will be no more potential difference, and no more work. The battery's voltage will be zero and it is said to be discharged. We've all seen this kind of action, if only by forgetting to turn off our car's headlights.

To understand figure one, we need to define some terms.

Charge is one of the basic properties of matter. It is a measure of one of the ways in which two pieces of matter exert forces on each other (gravity is a similar property). A quantity of electric charge is measured in "coulombs." One coulomb of charge is about 6.24E18 (6.24 times 10 to the eighteenth power) electrons. The charge on a single electron is -1.60E-19 coulomb.

The number of charges which flow between our battery terminals in one second is the "current", measured in amperes or milliamperes (thousandths of an ampere). One ampere of current is one coulomb of charge flowing in one second, or about 6,240,000,000,000,000,000 electrons per second.

The amount of current which flows through our resistor depends on the electrical force supplied by the battery. There must be an imbalance of charge or a potential difference between two points to sustain a current between them. The potential of the battery is called its "electromotive force", or emf. Electrical potential is defined in terms of work. Two points are at a potential difference of one volt if one joule of work is required to move one coulomb of charge between them. A joule is the amount of work performed when a force of one newton moves a point one meter (one joule = one newton-meter of work or energy).

The resistor in our circuit is not a perfect conductor; it

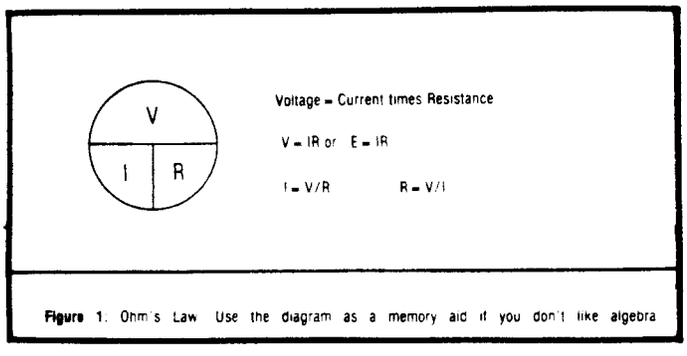


Figure 1: Ohm's Law. Use the diagram as a memory aid if you don't like algebra.

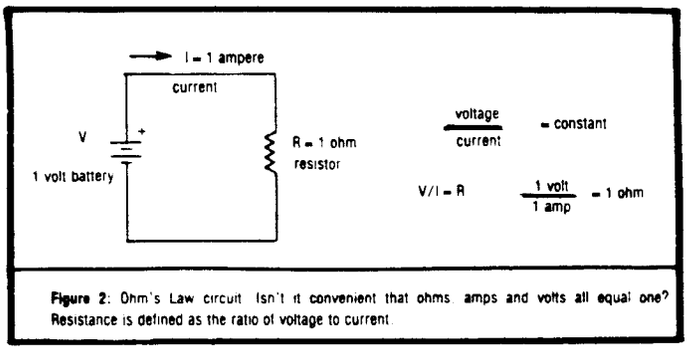


Figure 2: Ohm's Law circuit. Isn't it convenient that ohms, amps and volts all equal one? Resistance is defined as the ratio of voltage to current.

resists the flow of electrons to some degree. The amount of opposition to electron flow is the measure of the resistor's "resistance", measured in ohms. You can also look at a resistor as a conductor; its "conductance" is the reciprocal of its resistance. Conductance is measured in Mhos (yes, that's Ohms spelled backwords).

What George Simon Ohm (1787-1854) discovered was that if he connected the terminals of a battery (actually a chemical wet-cell) together using various kinds of conductors, the ratio of the voltage across the conductor to current through it was constant. That is, more voltage caused more current to flow. The ratio of voltage to current is a measure of the electrical resistance of the conducting material. One ohm is the electrical resistance when a potential difference of one volt causes a current flow of one ampere. This is what we now know as Ohm's Law.

Getting Practical

How much resistance does a resistor or other conducter offer? Connect the resistor to a battery, measure the voltage across the resistor and the current through it (see figure 3), then calculate the resistance through it with Ohm's Law: R = V ÷ I Then measure the resistor's resistance with your ohmmeter. You will find some error because the milliammeter itself has some resistance, so less current flows when the meter is in the circuit in series with the resistor. Additional error stems from the meter's limited accuracy and from less-than-perfect measuring technique. Try different resistors but don't try resistor values much lower than 500 ohms; they'll get too hot, and a 9-volt transistor-radio type battery can't supply enough current.

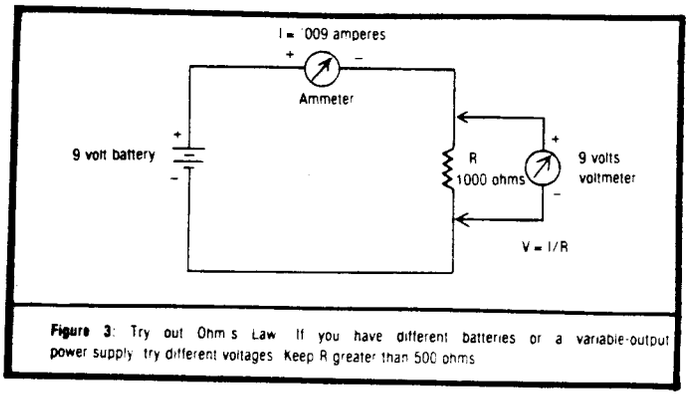


Figure 3: Try out Ohm's Law. If you have different batteries or a variable-output power supply try different voltages. Keep R greater than 500 ohms.

Ohm's Law says that if we increase the voltage, more current will flow. If we increase the resistance, less current will flow. Usually, we have a fixed voltage source and we control the current flow by varying the resistance.

Figures 4 and 5 illustrate the use of Ohm's Law. Knowing any two of the three parameters, we can calculate the unknown one. Give it a try.

Voltage Drops

Another way of looking at Ohm's Law shows that if we apply a voltage to a complex circuit, the current which flows through each resistance produces a "voltage drop" across

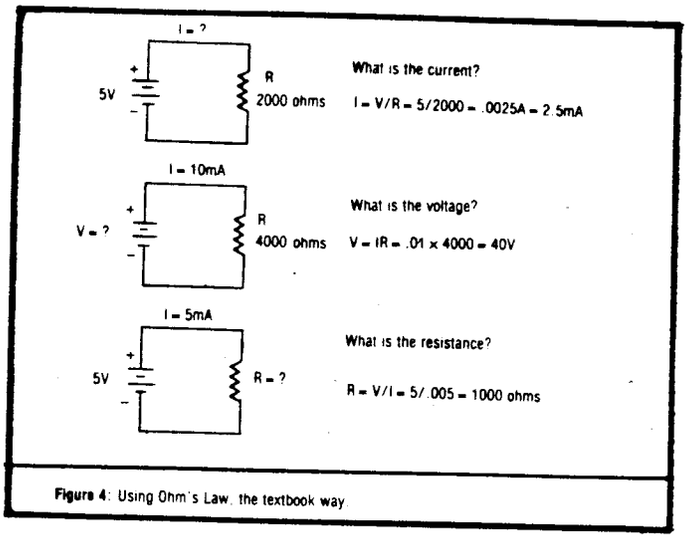


Figure 4: Using Ohm's Law, the textbook way.

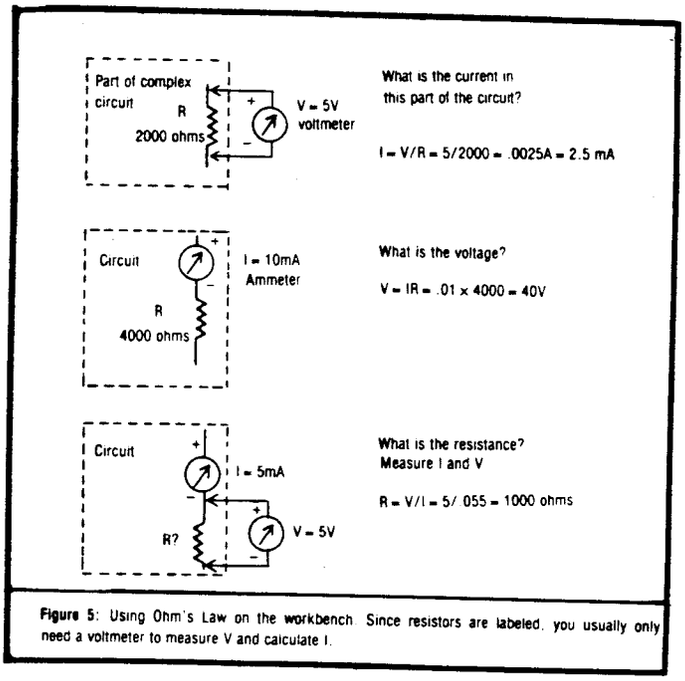


Figure 5: Using Ohm's Law on the workbench. Since resistors are labeled, you usually only need a voltmeter to measure V and calculate I.

the resistance equal to the product of current and resistance (see figure 6). More current produces a larger voltage drop. This may make more sense if we measure the voltage across each of two resistors connected in series (figure 7).

This figure contains a wealth of information. The total resistance of two resistors in series is the sum of the two resistance values. The same current flows through both resistors. The voltage drop across each resistor equals the current through it times its resistance. We have only one

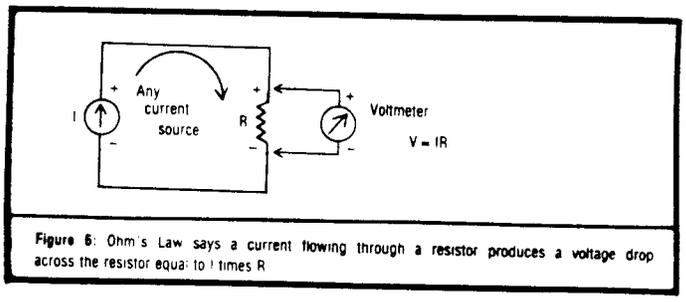


Figure 6: Ohm's Law says a current flowing through a resistor produces a voltage drop across the resistor equal to I times R.

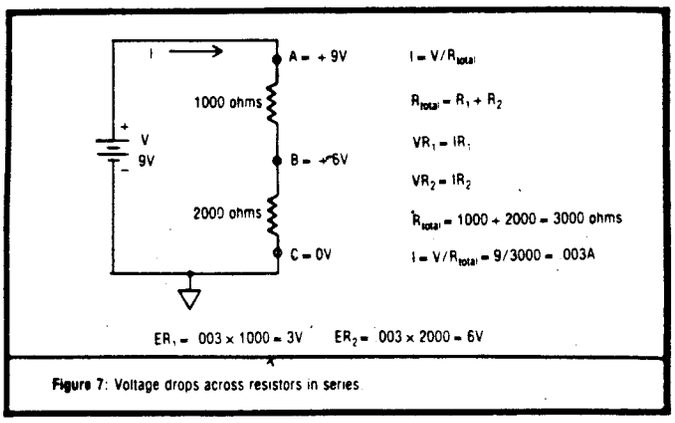


Figure 7: Voltage drops across resistors in series

current value (three milliamperes) but two voltages. If we take the most negative point in the circuit as a "reference" from which to make all voltage measurements (usually called "ground") then point A = 9 volts and point B = 6 volts. Do you see the reason for the expression "There is a three volt drop across R1 and a six volt drop across R2."?

A common convention is to use the most negative point in a complex circuit as a reference; then the most positive point has the "highest" potential or voltage. The voltage "drops" across series resistances until we reach zero or "ground."

While we're on the subject of conventions, there is sometimes confusion about the direction of current flow through a circuit. There are both negative and positive charges, and carriers of these charges. The two types of charges move in opposite directions when forced through a conductor by a voltage. In figure 8, electrons move from the battery's negative terminal, through the resistor and into the positive terminal. This is called electron flow. We will indicate the direction of current flow as "conventional current", in which current flows from a more positive to a less positive potential. It doesn't really matter which is used as long as we are consistent.

Power

To avoid having resistors go up in flame, you need to know how to calculate power dissipation. When a source of voltage pushes a current through a resistor, work is done, energy is used and heat is produced. How hot a resistor gets depends on its size and composition, but is proportional to the rate at which work is done moving charges through it

. One joule of work is done moving one coulomb of charge through a potential difference of one volt. Power is defined as the rate of doing work, in joules per second. We more commonly express electrical power in "watts". One watt of power (rate of doing work) is defined as one joule of work per second. Power in watts is calculated as volts times amperes:

Power (watts) = joules/second =
volts x coulombs/second = volts x amps.

$$P = V \times I$$

Figure 9 shows various ways of calculating power, found by using Ohm's Law and substition. Use these formulas on the earlier examples to find out the power in the resistors.

Resistors are manufactured in a wide variety of types and sizes. Their specifications are in ohms (resistance), accuracy or tolerance (%), temperature stability (ohms per degree Celcius), and power dissipation rating. The last parameter indicates how fast the resistor can get rid of the heat caused by a current moving through it. A one-watt resistor, for example, can safely handle the heat from one joule per second (one watt) if it is in open air at about room temperature. Unfortunately, some types of resistors suffer permanent changes in resistance if you get them too hot, even within their wattage ratings. Most resistors run at their rated wattage get hot enough to burn your fingers (especially high-power ceramic resistors). Carbon composition and some carbon film resistors can literally go up in flames if their wattage rating is exceeded.

Before you install a resistor in a circuit and turn on the power, you must calculate the expected power in the resistor, with any of the three formulas in figure 9. Then select a ¼ watt, ¼ watt, or larger size resistor. Most microcomputer circuits use a five volt power supply and very low currents; since the wattage needed is the product of voltage and current, you can usually use a ¼ watt resistor.

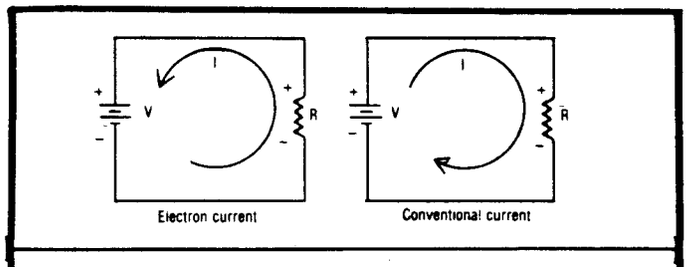


Figure 8: We will use conventional current flow from positive to negative.

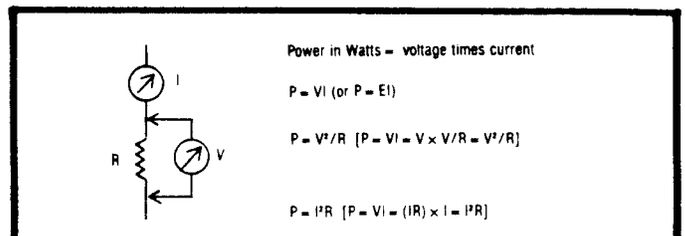


Figure 9: Formulas for calculating power in a resistor.

Resistors In Series And Parallel

You often won't have exactly the right value of resistor called for by your calculations. You can "dummy-up" an equivalent resistance by combining resistors in combination as shown in figure 10.

Adding a series resistor increases the total resistance. Adding a resistor in parallel decreases the total resisance. Notice that putting two equal resistors in parallel gives you an equivalent resistor of ½ of each resistor. Putting three in parallel divides by three.

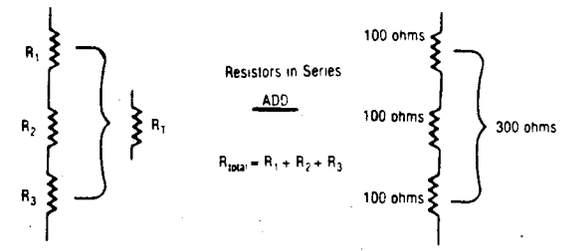


Figure 10a: Series resistances sum to an equivalent resistance

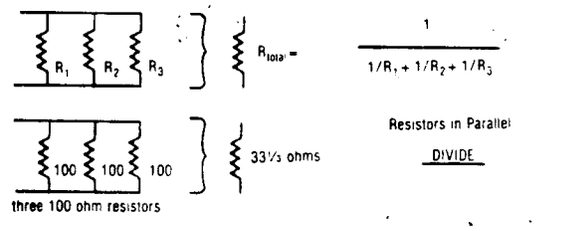
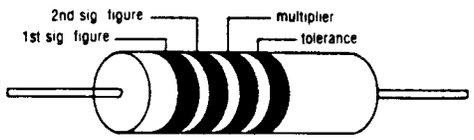


Figure 10b: Parallel resistances divide equivalent parallel resistance is the reciprocal of the sum of the reciprocals



Resistor Color Code

Color	Significant figure	Multiplier	Tolerance
Silver	—	.02	10%
Gold	—	.01	5%
Black	0	1	
Brown	1	10	
Red	2	100	
Orange	3	1000	
Yellow	4	10000	
Green	5	100000	
Blue	6	1000000	
Violet	7		
Grey	8		
White	9		
No Color	—		20%

Figure 11

Remember that the power dissipated in each resistor is the product of current through it times the voltage across it. When you make up an equivalent resistor, the power is spread among the individual resistors. You can make a high-wattage equivalent resistor out of a number of lower wattage ones by putting many higher-valued resistors in parallel, or lower-valued ones in series. For example, 10 resistors of 100 ohms, ½ watt each in parallel is the equivalent of one resistor of 10 ohms with a power rating of five watts. A series string of ten 10 ohm ½ watt resistors can handle 5 watts of power, but will have a resistance of 100 ohms.

Resistor Color Code

Carbon composition and carbon film resistors are marked with color bands as shown in figure 10. The four color bands are offset towards one end of the body of the resistor. To read the resistance value, hold the resistor end nearest the bands toward your left, then read the colors from left to right. The first two colors are the more significant digit and less significant digit of the resistance value. The third band is the multiplier, or number of zeros to tack on after the two significant digits. The fourth band indicates tolerance, or how far from the indicated resistance the specific resistor might be. Gold is 5%, silver 10%, and no band is 20%. A 1000 ohm, 5% tolerance resistor's real resistance can be anywhere from 950 to 1050 ohms.

Next Time

Next month we'll design a power supply. We'll go into how to select the components and calculate the required values for a five volt, 1.5 amp experimeter's bench supply. ■

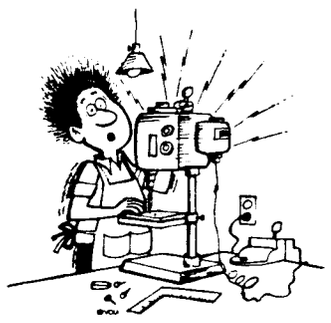
continued from p.18

ASCII Table

```
0E80- 20 20 45 54 49 41 4E 4D
0E88- 53 55 52 57 44 4B 47 4F
0E90- 48 56 46 20 4C 20 50 4A
0E98- 42 58 43 59 5A 51 20 20
0EA0- 35 34 20 33 20 20 20 32
0EA8- 20 20 20 20 20 20 20 31
0EB0- 36 3D 2F 20 20 20 20 20
0EB8- 37 20 20 20 38 20 39 30
0EC0- 20 20 20 20 20 20 20 20
0EC8- 20 20 20 20 3F 20 20 20
0ED0- 20 20 20 20 20 2E 20 20
0ED8- 20 20 20 20 20 20 20 20
0EE0- 20 20 20 20 20 20 20 20
0EE8- 20 20 20 20 20 20 20 20
0EF0- 20 20 20 2C FF FF 00 00
0EF8- FF FF 00 00 FF FF 00 00
```

■

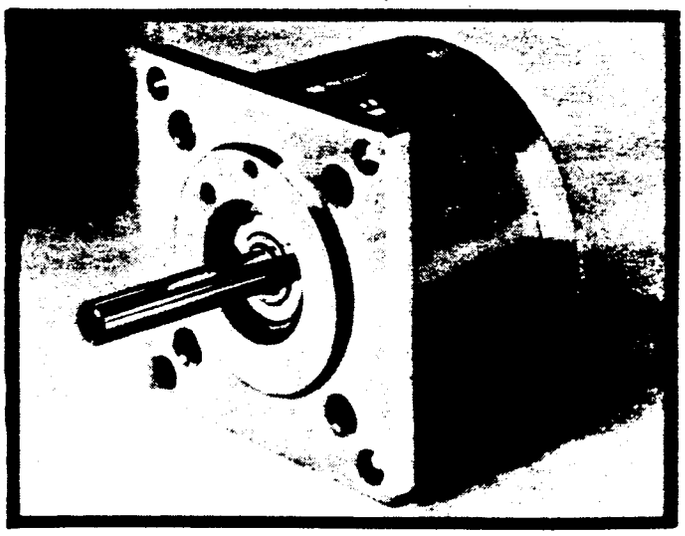
Any Computer Hacker Machinists Out There?



We want to contact anyone interested in using micros to control machine tools for personal use, or on a small business level. Contact us if you are using a micro for measurement or control with a lathe, milling machine, or other machine tool. We need to know what you are doing, how you are doing it, what problems you have, and what additional information would help you.

NEW PRODUCTS

Size 23 Stepper Motors from Clifton



Clifton Precision, Litton Systems, Inc., has expanded its line of size 23 1.8° stepper motors to include models with up to 170 oz-in holding torque and 120 oz-in dynamic torque. The motors operate with 200 steps per revolution.

The Clifton motors are compact; they are available in lengths as short as 1.5", excluding shaft. In addition, they are quiet operating, making them ideal for office environments. They are well suited to applications such as carriage wheels for matrix and daisy wheel printers, paper-feed drives, machine tool controls, disk drive head positioners, tape readers, plotters, robotic systems — wherever precise positioning is required in an open-loop system.

These size 23 stepper motors offer standard accuracy of ± 5 percent; accuracies of up to ± 3 percent are available. High reliability is an advantage of the Clifton motors, achieved through close-tolerance construction.

Special winding configurations, mounting, and other modifications can be produced to meet specific customer requirements.

For additional information on the Clifton size 23 stepper motors, contact Clifton Precision, P.O. Box 160, Murphy, NC 28906; (704) 837-5115, TWX 510-935-1068. ■

Ma Bell Hits Modems With Tariff

The September issue of ***The Computer Shopper*** *(P.O. Box F, Titusville, FL 32796), reported that Southwestern Bell Telephone Company's Oklahoma tariffs call for the charging of an 'Information Terminal Service' rate for anyone connecting a computer to the telephone lines via a modem.*

This rate is approximately 500% higher than the standard residential base rate. Obviously, this tariff dramatically affects the entire industry, as it practically prohibits noncommercial modem use.

The Computer Hacker considers easy access of communication lines thru modems to be very important for micro users, and we would like to publish a report of how this access is handled in different parts of the country.

If you have had an experience with modem connections that would be helpful to others in similar situations, please write and describe your experience and the solutions you arrived at. Include anything that you feel would be helpful to someone faced with a similar problem. For your protection, no names will be published, only the State will be given.

CLASSIFIED

Rate:$.50 per word, minimum charge $7.50. All classified ads must be paid in advance, and will be published in the next available issue. No checking copies or proofs are supplied.

WANTED: Teletype KSR-35 manuals needed to restore old teletype machine. Also need manuals for paper tape punch and reader. The Computer Hacker, P.O. Box 1697, Kalispell, MT 59903-1697.

FOR SALE: SSM 10/4 board for S-100 bus. Two serial, two parallel ports. $100. DEC LSI-11 minicomputer. Rack mount. KD11-F processor with KEV11 harware math chip, DLV11 serial card, DRV11 parallel card. Total of 48K RAM. Paper Tape O.S. $995. Write Lance Rose. c/o The Computer Hacker, Box 1697, Kalispell, MT 59903-1697.

Authors Wanted! We are interested in publishing specialized, well written, booklets for the serious computer user. There is often need for information which is too long or too specialized for a magazine, and too short for a major book. In order to publish this information in a magazine it is shortened and re-written for a broad general audience. Or, it is puffed up to fill a book. Neither of these approaches fills the need of the hacker. We will publish booklets of approx. 10,000 to 60,000 words, in addition to our magazine. If you have a manuscript which is too long for a magazine and too short for a major book, contact The Computer Hacker. Please query by letter with an outline and a self-addressed stamped envelope before sending your manuscript.

Advertise Where The Action Is

The Computer Hacker is THE place to advertise products for those who build, interface, and control with microprocessors. Send for our advertising media kit.

Advertising Department
The Computer Hacker
P.O. Box 1697, Kalispell, Mt 59903-1697